



Curso de Java Standard



Ing. Octavio Robleto



octavio.robleto@gmail.com



<https://octaviorobleto.com>



Introducción

La Sobreescritura de Miembros o Métodos, también conocida como Reescritura de Métodos es un concepto que se puede aplicar cuando se hereda en los Lenguajes Orientados a Objetos; cuando una subclase hereda todos los métodos accesibles de la clase padre puede redefinir el comportamiento de dichos métodos si así lo requiere.

Una subclase sobrescribe un método de su superclase cuando define un método con la misma firma del método de su padre, es decir, las mismas características (tipo de retorno, identificador, número y tipo de argumentos).




Sobreescritura (overriding methods)

La idea es poder agregar o cambiar el comportamiento de los métodos que posee el padre.

Se puede llegar a confundir el concepto de sobrecarga y sobreescritura, la diferencia radica que para sobrecargar un método hay que duplicar el nombre el método (en la misma clase) con el mismo tipo de retorno pero utilizar una lista de argumentos diferente, por otro lado, al sobreescribir un método no solamente el nombre y el tipo de retorno deben ser iguales, sino que ha de serlo también la lista de argumentos.

Tomemos como ejemplo el método `mostrarDatos` de la clase `Auto`.

```
public String mostrarDatos() {  
    String mensaje = "El Auto de la concesionaria " + concesionaria + " es de color " + color + ", marca " + marca  
        + ", patente " + patente + " y se encuentra " + ((encendido) ? "encendido" : "apagado");  
    return mensaje;  
}
```



Este método es heredado por las clases `Compacto` y `Camion` pero como es un método de la clase padre no posee los atributos adicionales de sus clases hijas, por lo que necesitaríamos crear un método adicional para cada clase con los atributos correspondientes.

¿Por qué sobrescribir?

La idea principal es extender la funcionalidad de los objetos sin tener que escribir métodos nuevos (nombres distintos pero con mismo fin) y así poder identificarlo correctamente.

Podemos sin ningún problema crear los métodos `mostrarDatosCamion` y `mostrarDatosCompacto` pero recordemos que la idea es aprovechar las herramientas de la POO ser lo más genérico posible y que sea de fácil entendimiento para cualquier desarrollador, por lo que dependiendo del objeto instanciado se entenderá correctamente que `mostrarDatos` devolverá una cadena de caracteres describiendo los atributos de dicho objeto.

```
@Override
public String mostrarDatos() {
    String mensaje = "El Compacto de la concesionaria " + concesionaria + " es de color " + getColor() + ", marca "
        + getMarca() + ", patente " + getPatente() + ", tiene " + puestos + " puestos, y se encuentra "
        + ((isEncendido()) ? "encendido" : "apagado");

    return mensaje;
}
```



```
@Override
public String mostrarDatos() {
    String mensaje = "El Camion de la concesionaria " + concesionaria + " es de color " + getColor() + ", marca "
        + getMarca() + ", patente " + getPatente() + ", tiene " + ejes + " ejes, "
        + ((disponible) ? "esta disponible" : "no esta disponible") + " y se encuentra "
        + ((isEncendido()) ? "encendido" : "apagado");

    return mensaje;
}
```



*Es común ver la notación **@Override** en los métodos sobrescritos pero no es obligatoria, siempre es recomendable colocarla para identificar fácilmente que el método se encuentra en una de nuestras clases padres y ha sido reescrito por el desarrollador.*

toString


Ya sabemos que a través de la herencia podemos disponer de los métodos y atributos de nuestros padres y también sabemos que por defecto nuestras clases heredan de la super clase **Object** por lo que tenemos disponible también sus métodos.

El método **toString** es un método de la clase **Object** que devuelve una cadena de caracteres con la representación de lo que es el objeto, símil a nuestro método `mostrarDatos`.

Por lo que a partir de este momento no tendremos mas nuestro método `mostrarDatos` y sobreescribiremos el método `toString` que representa la misma lógica.


```
@Override
public String toString() {
    String mensaje = "El Auto de la concesionaria " + concesionaria + " es de color " + color + ", marca " + marca
        + ", patente " + patente + " y se encuentra " + ((encendido) ? "encendido" : "apagado");

    return mensaje;
}
```




```
@Override
public String toString() {
    String mensaje = "El Compacto de la concesionaria " + concesionaria + " es de color " + getColor() + ", marca "
        + getMarca() + ", patente " + getPatente() + ", tiene " + puestos + " puestos, y se encuentra "
        + ((isEncendido()) ? "encendido" : "apagado");

    return mensaje;
}
```



```
@Override
public String toString() {
    String mensaje = "El Camion de la concesionaria " + concesionaria + " es de color " + getColor() + ", marca "
        + getMarca() + ", patente " + getPatente() + ", tiene " + ejes + " ejes, "
        + ((disponible) ? "esta disponible" : "no esta disponible") + " y se encuentra "
        + ((isEncendido()) ? "encendido" : "apagado");

    return mensaje;
}
```



toString

Cuando el método `toString` no se encuentra sobrescrito en nuestras clases en la cadena de herencia al invocar ese método te mostrará el nombre de la clase del objeto instanciado, el paquete al que pertenece seguido de un arroba y un número hexadecimal que representa la posición de memoria donde se encuentra dicho objeto: **com.educacionIT.javase.entidades.Camion@16f65612**

Es conveniente indicar que no es necesario invocar al método **toString**, es decir, colocar el nombre del objeto o colocar el nombre del objeto.`toString()` es equivalente, ambas formas invocan al **toString** mas cercano en la cadena de herencia.

```
Auto auto1 = new Auto("Plateado", "Audi", new Patente("ZBG-999", true), true);

System.out.println(auto1);
System.out.println(auto1.toString());
```

Al ser un método que reescribimos normalmente en todas nuestras clases, los IDE proporcionan un atajo para darle esta representación en cadena de caracteres: nombre de la clase y encerrado entre corchetes los atributos de la clase.


```
@Override
public String toString() {
    return "Auto [color=" + color + ", marca=" + marca + ", patente=" + patente + ", encendido=" + encendido + "]";
}
```

super


Esta palabra reservada hace referencia a su padre mas inmediato y la vimos cuando queremos utilizar en nuestro constructor el constructor de nuestro padre.

Pero también se puede utilizar para invocar un método de nuestro padre en la clase, muy usado en los **toString**.

```
@Override
public String toString() {
    return "Camion [toString()=" + super.toString() + ", ejes=" + ejes + ", disponible=" + disponible + "];"
}
```



```
@Override
public String toString() {
    return "Compacto [toString()=" + super.toString() + ", puestos=" + puestos + "];"
}
```



final

Un método puede tener la palabra reservada **final** cuando queremos evitar que las clases derivadas o clases hijas cambien la implementación o comportamiento (sobreescribir) de dicho método.

```
public final void guardar() {  
    System.out.println("Guardando Auto");  
}
```

Es mas común de lo pensado y en java tenemos muchos métodos finales y la clase **Object** no escapa de esto, dicha clase tiene varios métodos finales que java no quiere que redefinamos como lo podemos hacer con el **toString**.