

Comunidad de objetos

Comunidad de objetos	1
¿Qué aprenderás?	2
Introducción	2
Sobrecarga de métodos	3
La sobrecarga de toString()	3
Ejercicio guiado: Vehículo	3
Las variables de instancia y las variables locales	6
Variables de clase	7
Ejercicio propuesto (1)	8



¡Comencemos!

¿Qué aprenderás?

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

Introducción

En este capítulo seguiremos viendo algunas definiciones de la Programación Orientada a Objetos, como por ejemplo la sobrecarga de métodos, el método `toString()`, las variables locales, las variables de instancia y las de clase. Además, veremos cómo se generan cada uno de estos conceptos, el contexto donde actúan y la forma en que podremos imprimir el estado de una instancia.

¡Vamos con todo!



Sobrecarga de métodos

Hay ocasiones en que necesitamos que un método reciba una lista de parámetros diferente. Cuando se presentan estos casos, lo que deberíamos hacer es sobrecargar un método, esto permite crear varias versiones (ya que se utiliza el mismo nombre de método, pero se cambian los parámetros). De esta forma 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()`, este es un método que 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.

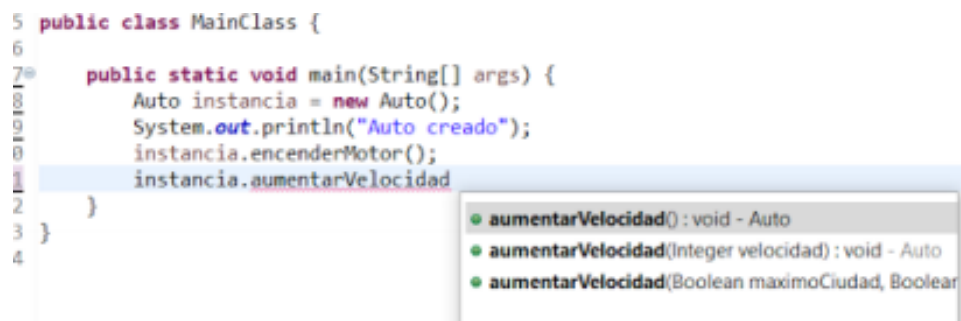
```
public class Auto(){
    private String marca;
    private String modelo;
    private String color;
    private int velocidadActual;
    private boolean motorEncendido;
}
```

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;
}
```

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;
    }
}
```



```
5 public class MainClass {
6
7     public static void main(String[] args) {
8         Auto instancia = new Auto();
9         System.out.println("Auto creado");
10        instancia.encenderMotor();
11        instancia.aumentarVelocidad
12    }
13 }
14
```

- `aumentarVelocidad(): void - Auto`
- `aumentarVelocidad(Integer velocidad): void - Auto`
- `aumentarVelocidad(Boolean maximoCiudad, Boolean maximoCarretera): void - Auto`

Imagen 1. Llamar al método "aumentarVelocidad" con sobrecarga.
Fuente: Desafío Latam

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]
```

Vemos que la instancia `Auto.toString()` es igual a "Modelo.Auto@15db9742" que es un valor que representa la instancia actual del Auto. Esto es muy difícil de traducir al español, así que vamos a crear una sobrescritura del método `toString` para que nos devuelva otro valor que podamos entender.

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.

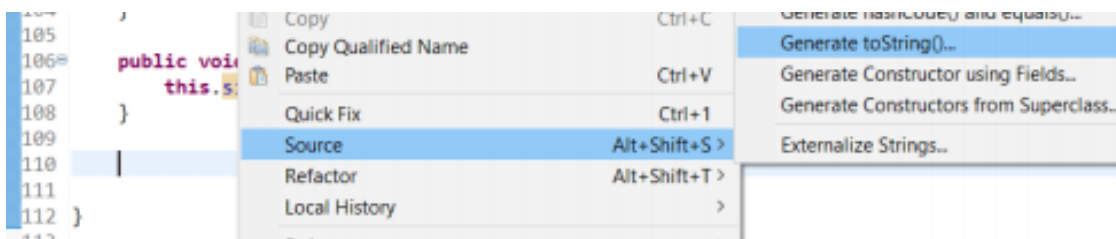


Imagen 2. Generate toString().

Fuente: Desafío Latam.

En el cuadro de diálogo aparecerán las variables para generar y las seleccionamos todas. Se generará el siguiente código.

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

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,
]
```

Se agregó la línea `@Override` por encima de la sobrecarga del método `toString()`. Esto es una anotación de sobrescritura, la diferencia entre esto y la sobrecarga es que este concepto se aplica cuando se quiere reutilizar métodos heredados de superclases.

Las variables de instancia y las variables locales

Las variables de instancia son aquellas que se mantienen "almacenadas" dentro de las instancias. Las variables locales son aquellas que se declaran dentro de los métodos y, al terminar la ejecución del método, se descartan.

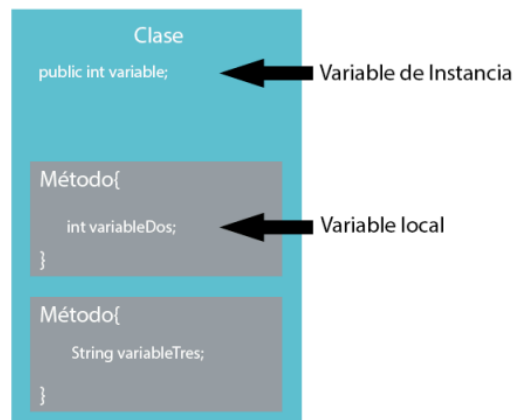


Imagen 3. Variables de instancia y variables locales.
Fuente: Desafío Latam.

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.

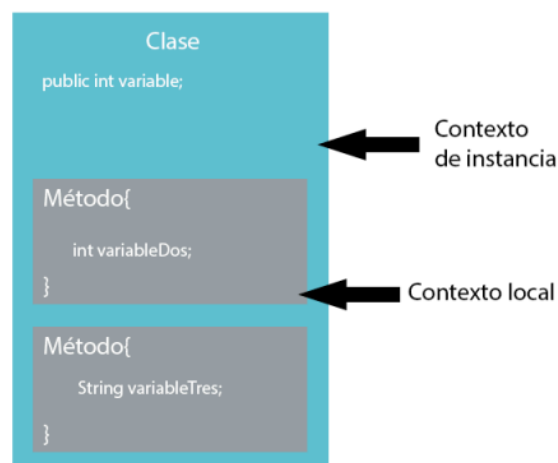


Imagen 4. Variables en el contexto.
Fuente: Desafío Latam.

Variables de clase

Dentro de la Programación Orientada a Objetos, así como existe un contexto de instancia, existe un contexto de clase (también conocido como estático). Este 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.

Así como se pueden crear variables de clase, se pueden crear métodos de clase, que 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. Lo mejor es crear métodos de clase (también conocidos como estáticos), para no tener que crear instancias de clases que utilizaremos solo para llamar a uno de sus métodos.

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.