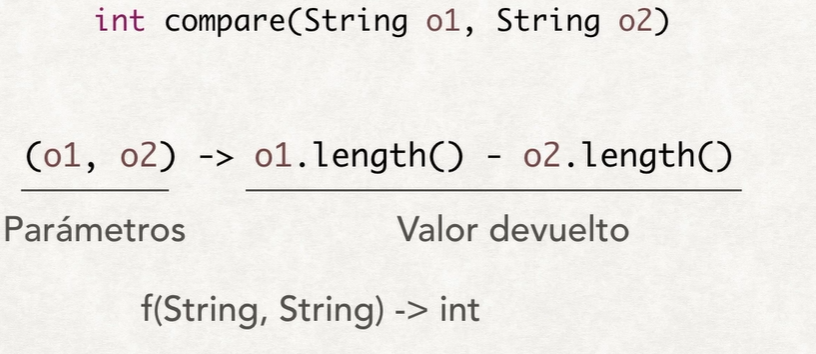
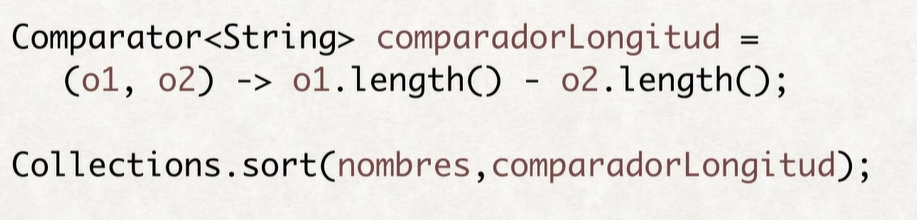
# **Java Avanzado Programación Funcional**

## **Introducción**

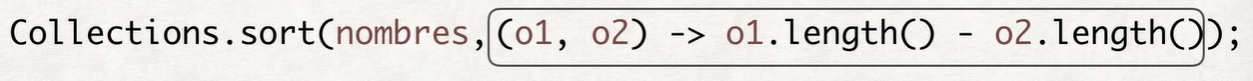
Lambda expresión tiene unos parámetros de entrada y un valor devuelto, por ejemplo:



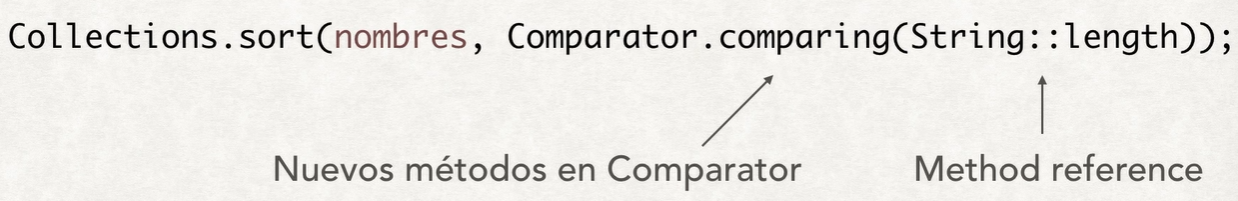


Las lambdas expressions permite evitar la encapsulación del comparator en una nueva clase.

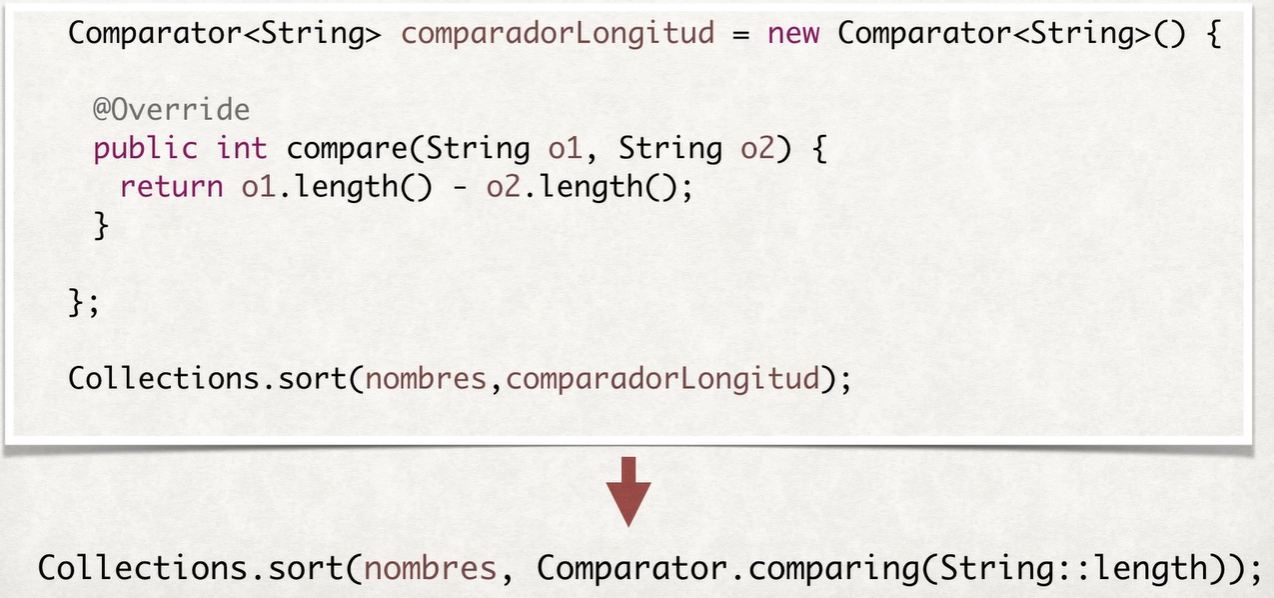
Este código puedo optimizarlo aún más utilizando la lambda expresión directamente como parámetro del Collections.sort:



También en la versión 8 de Java se han ampliado los tipos del JDK que ya conocemos para sacar ventaja de las lambdas expresión, por ejemplo, podemos utilizar nuevos métodos y referenciar métodos ya existentes de una clase:



Como resultado veremos la diferencia de cómo se haría la comparación en Java sin lambda expressions y Java 8 con lambda expressions:



Con Java 8 logramos mejorar nuestro código haciéndolo más compacto, expresivo y reutilizable.

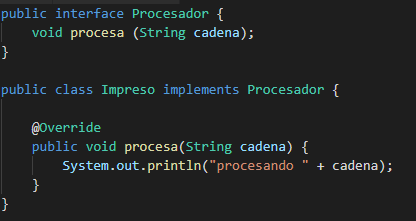
## **Interfaces**

### **Default Methods**

Es una de las novedades introducidas en Java 8, recordando la definición de interface es aquella que permite definir que aspectos tendrán un conjunto de clases de nuestra aplicación sin definir su comportamiento exacto.

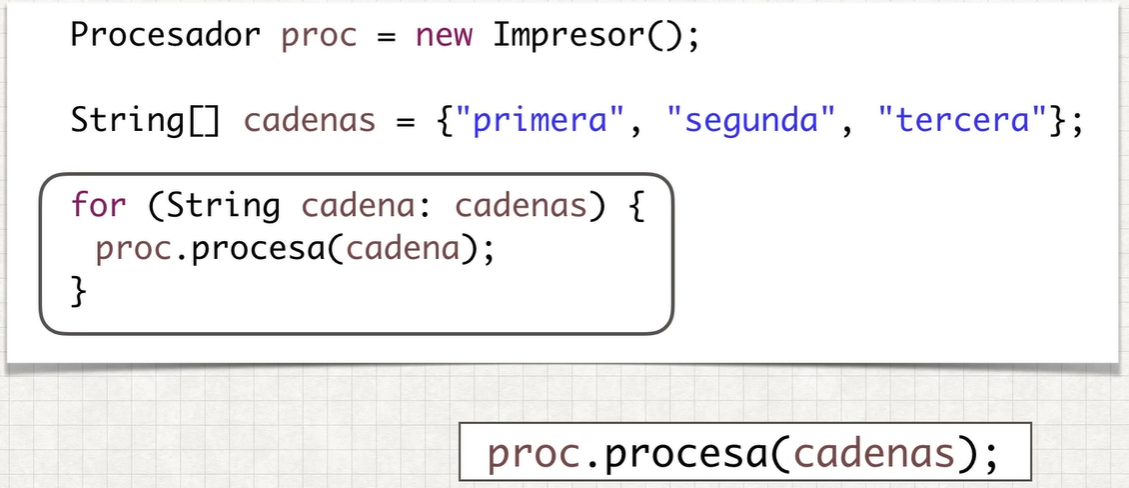
Las mismas son implementadas por distintas clases que dan una implementación especifica el método correspondiente.

Por ejemplo:

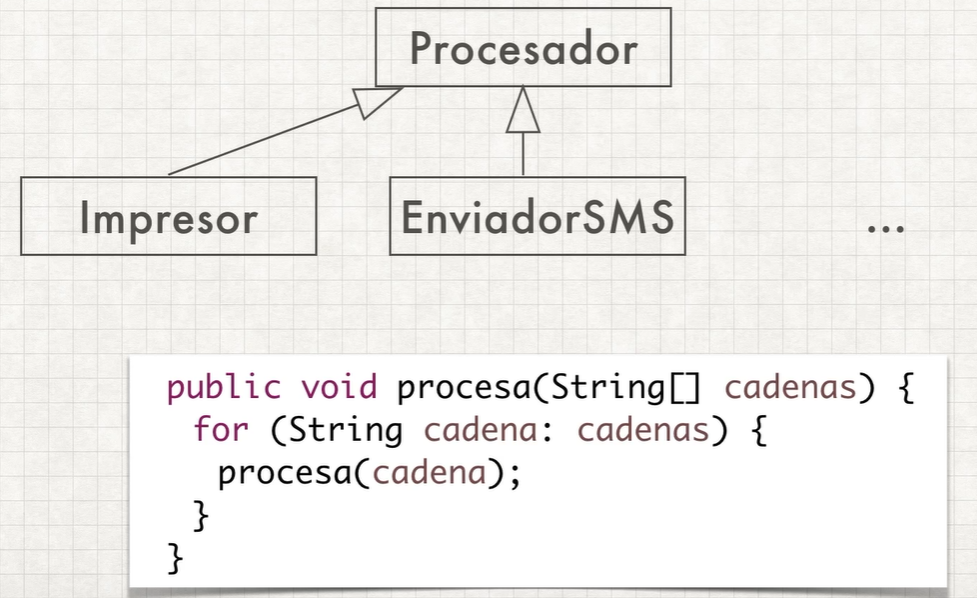


* Antes de Java 8 las interfaces no podían disponer de ninguna implementación de métodos.
* A partir de java 8 podemos implementar métodos default en las interfaces, lo cuál es útil debido al limitante de Java de no proveer herencia múltiple y para definir un comportamiento común para varios clientes de una API.

Por ejemplo, tenemos varios clientes que utilizan la interface procesa en un bucle for, por lo que cada vez que esta necesidad aparece, se repite el mismo código, por lo que quisiéramos que el cliente llame a procesa con un array de cadenas y que la lógica se defina dentro de dicho método:



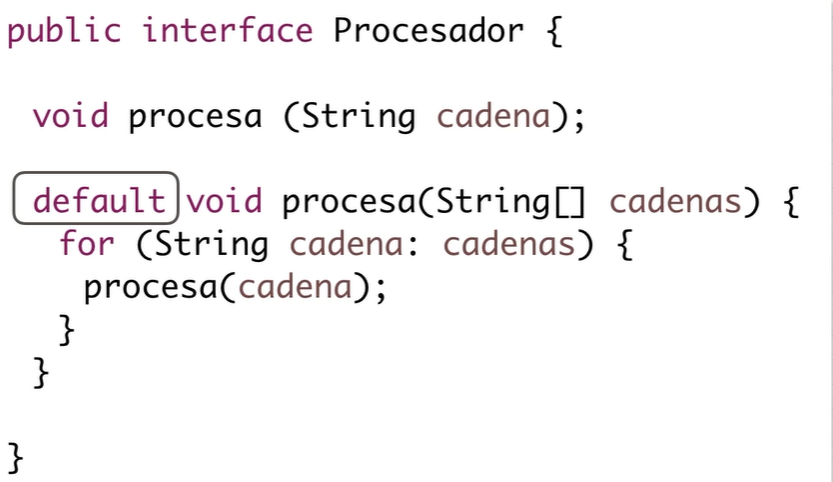
Una forma clásica de hacerlo es añadir el método que queremos en la interface, pero esto nos genera un problema de que el mismo deberá ser implementado en todas las clases que implementen la interface.



Si hay muchas clases que implementan la interface, cada una de ellas deberá implementar el método procesa(String[] cadenas) y su lógica siempre será idéntica, recorrer el array y ejecutar el método procesa de un solo String, el cuál si es peculiar a cada una de las implementaciones diferentes.

**¿Por qué debe cada subclase implementar esta versión del método procesa con un array de String?**

* Antes de Java 8, esta situación se resolvía con clases abstractas que permiten combinar métodos con implementación, con los llamado métodos abstractos, pero como hemos mencionado utilizar clases abstractas no es tan conveniente como utilizar interfaces.
* En Java 8 se han introducido los **default methods** añadiendo la palabra clave default a un método podemos proporcionar en la misma interface una implementación por defecto a un método:



Este método puede ser sobrescrito por cualquier clase que implementa la interface si así se considera necesario.

Por otra parte, tampoco hay ninguna limitación en usar desde los default methods métodos que no están implementados en la interface.

Gracias a los default methods las clases que implementen dicha interface solo deberán implementar los métodos abstractos a la misma, es decir los que son particulares para cada implementación, no debe implementar los métodos para los cuales la interface ya tiene toda la información para complementarlos.

De esta forma el cliente puede usar indistintamente los métodos procesa con un String o con un String[].

#### **Características**

* Una interface puede tener cualquier número de métodos default.
* Las clases que implementa la interface puede sobre-escribir los métodos default.
* Si una clase implementa dos interfaces y recibe dos métodos default con la misma forma, es obligatorio sobrescribir el método.

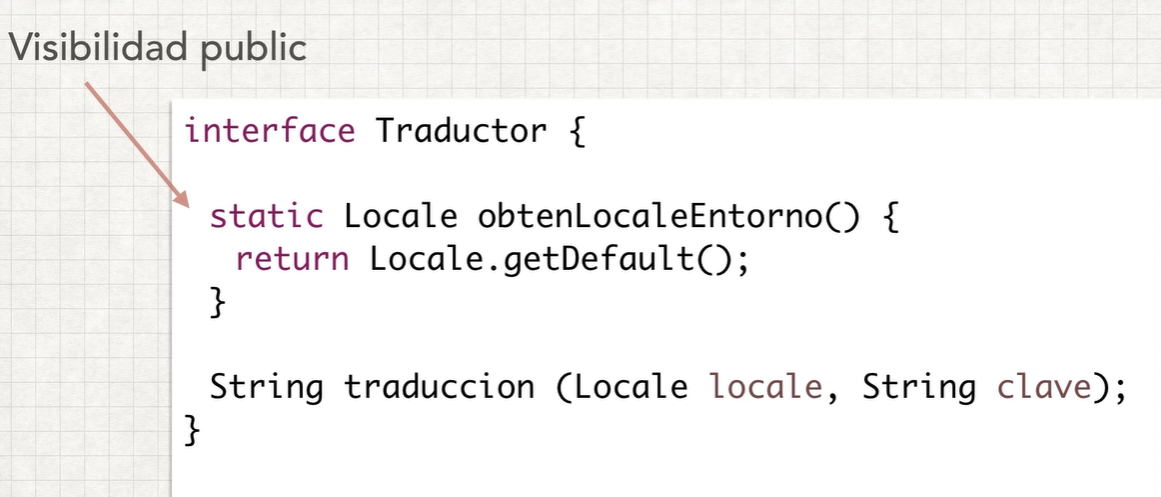
### **Static Methods**

Otro de los cambios introducidos en las interfaces en Java 8 es permitir la definición de métodos estáticos, los cuáles eran exclusivos de las clases.

Este cambio permite introducir implementación en la definición de una interface, como en las clases los métodos estáticos no pueden referenciar a ningún método o variable no estáticos, sirven para proporcionar funcionalidad que no depende sobre que instancia se ejecuta el método.

Un ejemplo podría ser una interface que sirve para obtener traducciones de mensajes a partir de una clave y del Local al queremos traducirlo.

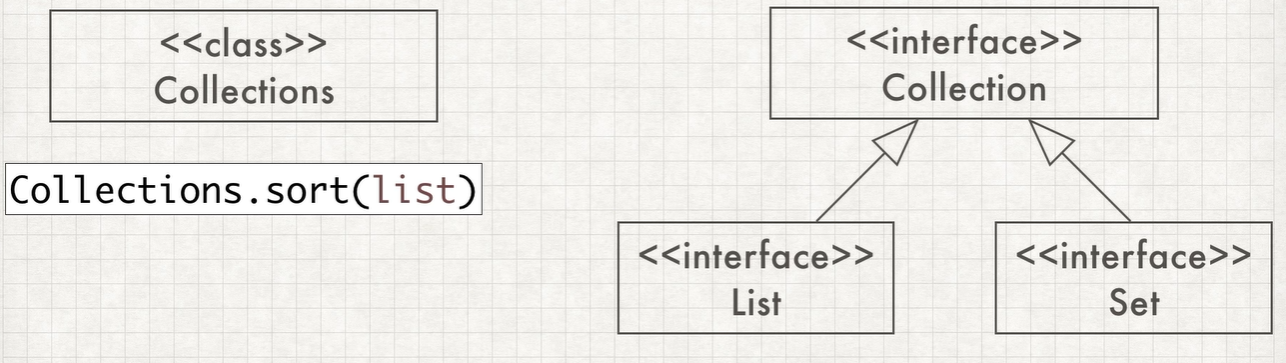
Para hacer más prácticas la interface podemos añadir un método que obtuviese el Locale por defecto del entorno sobre el que se ejecuta la aplicación:



En versiones anteriores a Java 8 antes de poder incluir métodos static en las interfaces se debía resolver de una manera no muy elegante.

Por ejemplo, el diseño del framework Collection, el cuál esta liderado por la interfaz Collection que describe que métodos serán comunes a todas la estructuras de datos que la implementen. A partir de esta se van definiendo subinterfaces más especificas y después numerosas implementaciones con distintas características.

Este framework además dispone de distintos métodos de utilidades generales no específicos a ninguna instancia, al no poder incluir estos métodos en la interface Collection, se crea una clase Collections y se ubican allí estos métodos. Collections dispone de un amplio catalogo de métodos estáticos como por ejemplo el método sort que sirve para ordenar una lista de elementos.



**Estos tipos de construcciones con los métodos estáticos en las interfaces ya no son necesarios.**