Tarea: Tema 7.

Reto FINAL – API Stream.

|  |
| --- |
| **Profesor:** Javier Puche  **alumno:** Arancha Chicharro Montejano  Fecha: 17/04/2023 |

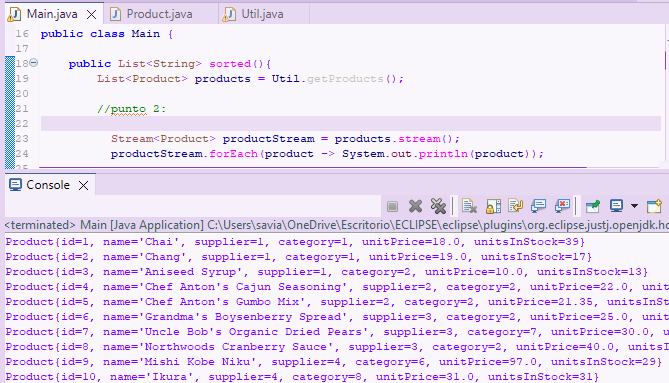
***TUTORIAL 1:***

**1. INTRODUCCIÓN:**

Después de importar el proyecto en Eclipse y de entender el documento, pasamos a realizar cada paso.

**2. INTERFAZ STREAM:**

Pruebo el código indicado para recuperar, mostrar el Stream, secuencia, de los productos de nuestra lista. Se puede ver que se muestra por pantalla todo el contenido del archivo productos.csv, de la lista.

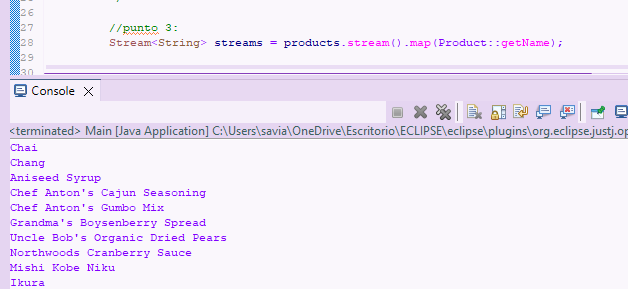
**

**3. CONSULTAS SIMPLES:**

Realizamos una consulta simple de sql pero usando Streams en java.

Primero, usamos el método stream para obtener la secuencia de elementos del tipo indicado, Product en nuestro caso. Es como decir de dónde queremos sacar los datos, esto es lo más importante.

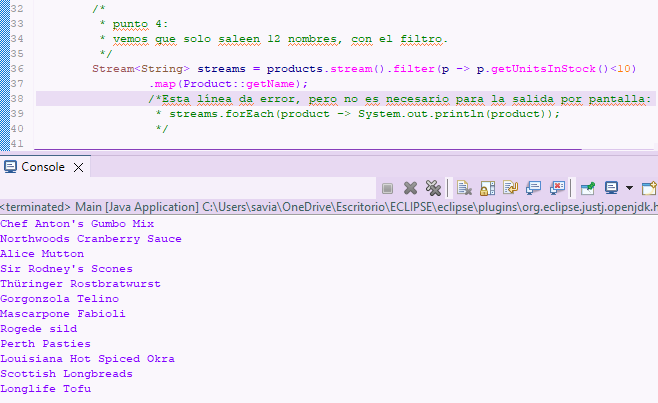
Y después usamos el método map para indicar qué atributo de nuestra lista queremos mostrar, en este caso solo mostramos el name.



**4. FILTRADO**

Aquí hemos hecho otra consulta en nuestra lista de productos pero esta vez con filtro.

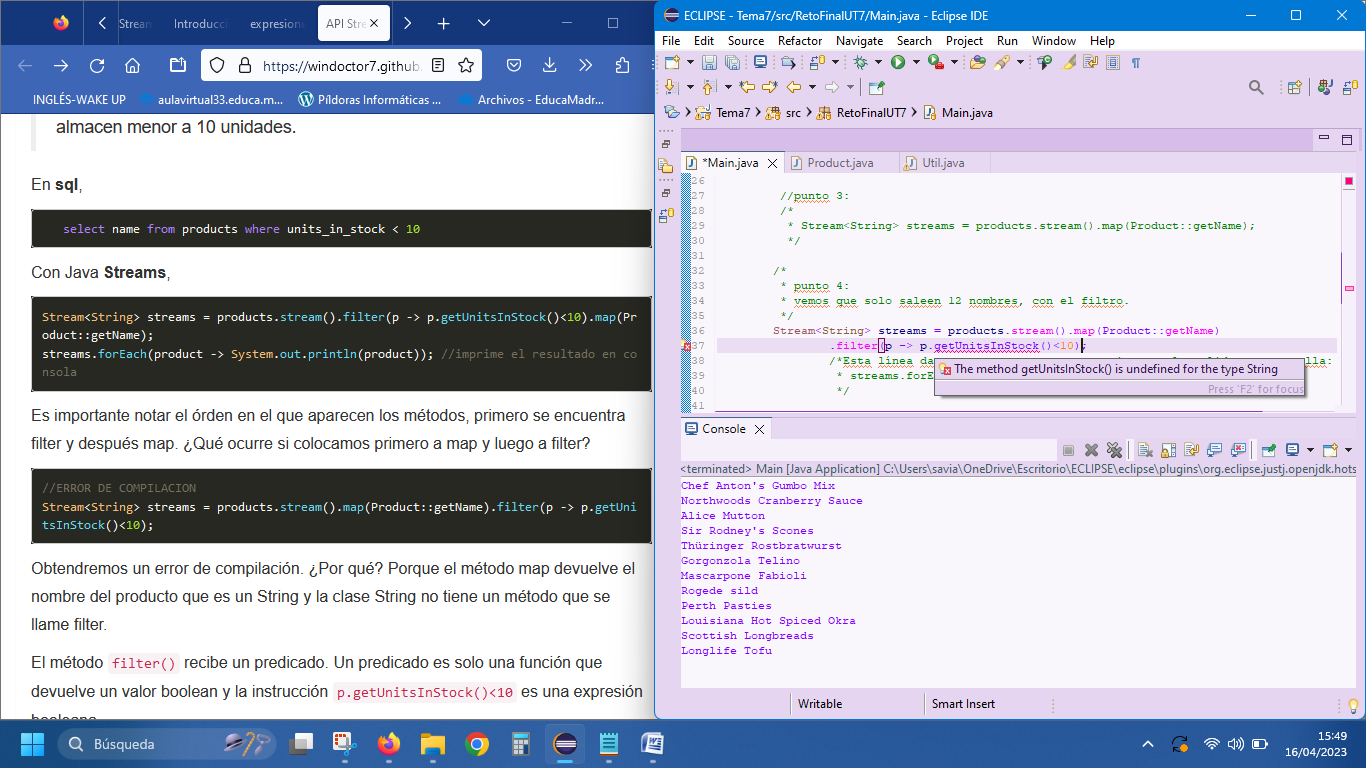
Sería usar el mismo código que el punto anterior pero añadiendo más cosas, en este caso añadimos el filtro de que solo se muestren los productos que en el campo *unitsInStock* sea menos de 10, y se hace añadiendo el método filter e indicando el campo por el queremos filtrar la info.



Aquí aprendemos de la importancia del orden en el código. Para que no de error de compilación, después del .stream, se pone el .filter y por último el .map.

Da error porque estamos diciendo que queremos un tipo String, el método .map devuelve el name que es tipo String, y el método .filter recibe una función que devuelve un boolean.

Comprobamos el error:



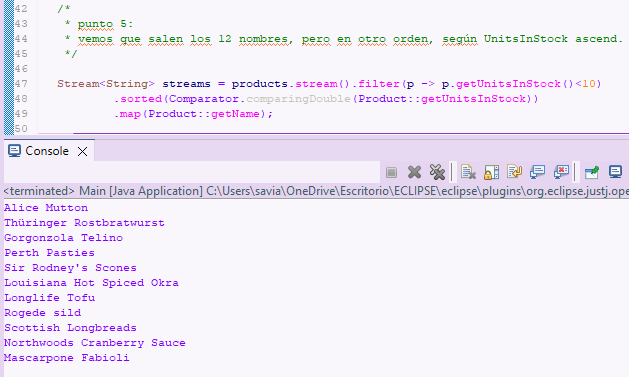
En lugar de expresión lambda, se puede usar una función.

**5. ORDENACIÓN.**

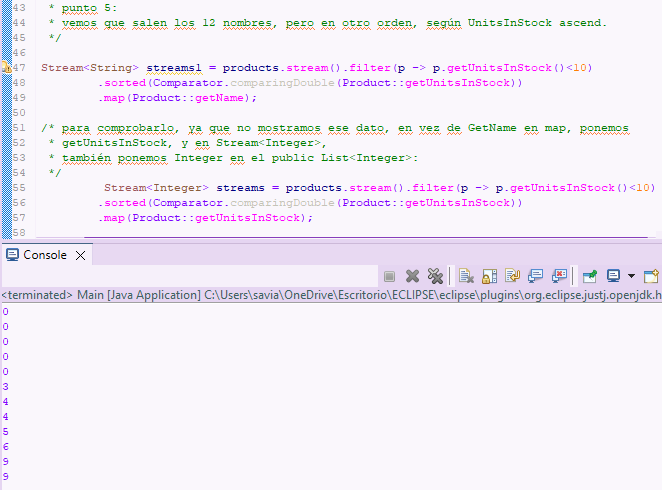
En este paso lo que hacemos es añadir al código y filtro anterior, un método de ordenación para que nuestra lista salga ordenada según el parámetro que indiquemos.

Se hace usando el método .sorted que recibe la interfaz *comparator*., que a su vez tiene algunos métodos que según el dato que pasemos por parámetro, podemos implementar (comparing con int, double o long)

*Ejemplo:* siguiendo con el código anterior, añadimos un orden de forma ascendente según el campo *unitsInStock,* y este método se coloca después del filtro y antes del map.



Para comprobarlo, ya que no mostramos ese dato, lo que he hecho es en vez de GetName en map, ponemos getUnitsInStock, y en Stream<Integer>, también ponemos Integer en el public List<Integer>. De esta manera muestro la misma lista, los mismos productos pero mostrando el mismo campo por el que lo estoy filtrando y así comprobamos que la lista sale ordenada de forma ascendente.



Para mostrar la lista ordenada de forma descendente, aplicamos un .reversed dentro del .sorted, seguidamente del Comparator.

