Práctica 5 ADSOF

Pablo Sánchez y Álvaro Rodriguez

1- Para el apartado 1 implementamos la lista ordenada sin mucha complicación, la implementación más complicada fue el método *addCriterion* ya que no sabíamos muy bien cómo hacer que recibiera un comparador de una clase que se pudiera comparar, hasta que finalmente lo arreglamos creando un tipo T => `T extends Comparable<T>`.

2- Este apartado no resultó demasiado complicado, implementamos una clase interna llamada `Line` , la cual obtenía una serie de expresiones lambda y una string que sería posteriormente modificada.
Luego el método emit llamaría a estas Lines por cada persona y crearía un mapa que retornaría para imprimirlo por pantalla (este mapa los implementamos tipo likedHashMap, para que luego al recorrerlo e imprimirlo por pantalla, las entradas se mostraran en el orden correcto según se iban añadiendo).
La clase de Person se ha implementado con los campos y métodos que se esperan en los tests.

3- Este apartado fue uno de los más complejos, ya que algunas expresiones lambda devolvían un tipo no igual al tipo de la entrada, por lo que tuvimos que arreglarnos para usar otro parámetro R que al final en Template se implementa como `? extends Object` para no romper con la condición de que Template solo parametriza R.
Para este apartado creamos dos clases más dentro de Template: `Predicates` que guarda un predicador y una string, y `Lambdas` que guarda una función que retorna un array de objetos parametrizados como R, un string y un array de expresiones lambdas como hacia `Line`.

4- Para este apartado, creamos una serie de clases para todas las estrategias a implementar, y una clase abstracta de la que heredan (`Strategy`).
Para realizar las estrategias de forma genérica (con la posibilidad de añadir más estrategias de forma genérica) hicimos que obtuvieron como parámetro la string que estaba tratando el flujo de estrategia y la persona que estaba siendo procesada.
De esta forma, la salida de un elemento de la estrategia se convertía en el de la siguiente, de forma que cualquier otra estrategia podría ser añadida a continuación o entre cualquiera de las anteriores.
Cabe mencionar que, para estrategia de FilePersister, a la hora de generar archivos .txt de cada persona, usamos un hashMap para almacenar contadores referentes a los nombres de objetos tipo Person, de esta forma, cuando se intenta procesar dos personas con el mismo nobre por esta estrategia, se generará un archivo con un numero mayor en 1 que el anterior (como pide en el enunciado).
Estas estrategias son llamadas en la rutina del método `emit(T p)`.

Diagrama de clases: