Informe sobre la experiencia de realizar los cambios

Funcionalidades:

Al querer agregar las funcionalidades que se solicitaron en esta segunda etapa del trabajo práctico nos encontramos con algunos problemas.
Para empezar tuvimos que comprender la lógica de la solución planteada por nuestros compañeros, fue necesario analizar cada una de las clases, y ver cómo se fueron distribuyendo las responsabilidades de cada tarea en cada clase.
Para ello se observaron métodos y atributos de cada clase, ademas se corrieron los test y se observó que probaban en cada caso, se notó que el código no estaba bien indentado lo cual dificulta la lectura.
Las responsabilidades no estan muy bien definidas, en la clase Procesador habia lógica que no estaba testeada y al ser una clase que contiene el Main si no se ejecutaba la aplicación no se podría probar dicha lógica También notamos que la clase GestorDeArchivos se encarga de: procesar los archivos CSV, obtener la lista de bicicletas, escribir el YML de salida, mover los zip procesados, filtrar y obtener archivos zip.
Surgieron dudas respecto de la clase Bicicleta la cual tiene un atributo Recorrido, lo cual nos pareció que no era correcto. Consideramos que una bicicleta puede tener más de un recorrido, aún así, nos adaptamos al código y seguimos esa línea.
Se agregaron test en la clase InformacionEstadisticaTest, para probar las nuevas funcionalidades, lo cual llevó a agregar la funcionalidad de mostrar cuál fue la bicicleta usada más tiempo. También fue necesario identificar que clase es la encargada de generar la información estadística, y observamos que la clase InformacionEstadistica posee las listas de las bicicletas más usadas, las bicicletas menos usadas, los recorridos más realizados y el tiempo promedio, por esa razón nos pareció correcto agregar en esta clase los atributos bicicletasUsadasMasTiempo, tiempoDeLaBicicletaMasUsada y tiempoDeProcesamiento, con sus respectivos get y set.
También se agregó el metodo generarBicicletasUsadasMasTiempo el cual recibe como parámetro un mapa con bicicletas y tiempo de uso, el cual es procesado y se genera la lista de aquellas usadas más tiempo.
En cuanto a la obtención de los datos de los archivos csv, notamos que la clase GestorDeArchivos, por cada registro que se lee genera una bicicleta, con su respectivo recorrido y luego las devuelve en una lista de bicicletas.
Esta lista de bicicletas es la que recibe por parámetro el generador de estadística y con esos datos genera información que luego va a ser utilizada por la clase InformacionEstadistica para obtener los datos solicitados.
Por este motivo es que la clase GeneradorDeEstadistica es la encargada de generar los tiempos de uso de cada bicicleta asi como tambien el tiempo de la bicicleta más usada, para ello se agregaron los métodos, almacenarTiempoDeUsoDeBicicletas y el metodo calcularTiempoDeBicicletaMasUsada. Ademas se tuvo que modificar el metodo generarEstadistica, donde se setea el tiempo de la bicicleta más usada y se genera el mapa de bicicletas y tiempo de uso que luego es utilizado por el atributo estadística para generar la lista de bicicletas usadas más tiempo. Se agregaron test a la clase GeneradorDeEstadistica para probar que los datos generados sean los correctos.
También se modificó la salida de los archivos yml, para que muestre los nuevos datos generados. Para

ello alteró el método escribirYML de la clase GestorDeArchivos, y se agrego el metodo escribirIdsBicicletasUsadasMasTiempo el cual va escribiendo los diferentes ID de cada bicicleta usada más tiempo.
La experiencia de la modificación fue buena, no hubo que tocar demasiado el código, los cambios se concentraron en 2 clases específicas, el GeneradorDeEstadistica y la clase InformacionEstadistica. Solo hubo que agregar algunos atributos y un par de métodos. Al hacerlo no interferimos en la ejecución de los demás test los cuales siguieron dando verde como antes de las modificaciones.
Lo que más costó fue entender bien la lógica del código y ver que responsabilidades tiene cada clase, una vez determinado esto fue más facil realizar las modificaciones de las funcionalidades.

Optimización:
Nos costó entender cómo estaba hecha la solución original por lo cual costó buscar una solución más óptima.

Se tomaron pruebas de tiempo de procesamiento del archivo zip:

Procesador: Intel Core i5-2410M CPU 2.30GHz × 4.
Memoria: 6GB.
Archivo zip: 88,1 MB.
Archivos internos: 12 csv de 37.4 MB cada uno, total: 448,8 MB.

Tiempo original:
Tiempo promedio de procesamiento: 13.18774 segundos

Solución 1:
Tiempo promedio de procesamiento: 22.31970 segundos

La solución planteada consiste en partir el stream de datos y canalizarlo por diferentes thread dependiendo de los procesadores de la máquina donde se corre la aplicación. Pensamos que esto mejoraría el rendimiento de la aplicación, pero no ocurrió. Luego de realizar las pruebas notamos un incremento del 65% de procesamiento.

Solución 2:
Tiempo promedio de procesamiento: 32.15105 segundos

Una segunda implementación fue cargar un archivo por vez en memoria y procesarlo paralelamente pero el tiempo promedio de procesamiento aumentó un 100%.
Solución 3:
Tiempo promedio de procesamiento: 13.13814 segundos

Otra fue procesar distintas cantidades de bicicletas con la solución original y el tiempo fue el mismo.

Luego de probar las distintas soluciones, analizamos los tiempos promedios y concluimos que al cargar de a un archivo en memoria y procesarlo de a 500 registros es mucho más rápido que leer todo archivo en paralelo y procesarlo.