# Untref – AyDOO – 2015

# Trabajo Final Parte II


## Alumnos: Ignacio Luna Echechuri (22763) y
## Martin Fernandez Gamen (22764)


# *Informe de cambios*

Para realizar las modificaciones necesarias al aplicar las nuevas funcionalidades, analizamos el código para evaluar cuales iban a ser las clases afectadas por los cambios.

Al hacer la evaluación y una ejecución del código, vimos que tenia graves problemas de performance al punto de que al cargar un archivo .zip de 14 mb, el tiempo de procesamiento era superior a los 10 minutos. Esto se debe a que los archivos, son completamente subidos a memoria donde al ejecutarlo en un cpu con 2gb de memoria y procesador Intel core 2 duo lanzaba el siguiente error: java.lang.OutOfMemoryError: Java heap space.

Luego de esto se puso en evidencia que antes de aplicar los cambios para que realice las nuevas funcionalidades, nos dedicamos a optimizar la performance del software.

***Para ellos hicimos los siguientes cambios:***

Para cada registro del archivo procesado se agregaba un nodo a una List<Travel>. Cada nodo contenía un objeto Bike, uno User, dos Location y uno Travel que contenía al resto.

Para solucionar el inconveniente de performance se modificaron las estructuras de datos en las cuales se almacenaba la información. Para ello se realizo lo siguiente:

Se generaron dos Hashmap:
**HashMap<Bike, TimeAndQuantityBike> mapBike;**
**HashMap<Travel, Integer> mapTravel;**

Al recorrer los archivos solo se guardan los registros nuevos en cada uno de los map, y al encontrar tanto Bike como Travels que fueron cargadas en los mismos, solo se modifican los contadores correspondientes, pero no se agregan como nuevos elementos.

De esta manera bajamos considerablemente la cantidad de objetos que se encuentran en memoria y además evitamos tener redundancia en la información almacenada.

Después de los cambios en optimizar la performance obtuvimos el siguiente resultado:
El procesamiento de un archivo .zip de 14 Mb se obtuvo la siguiente salida con su respectivo tiempo de procesamiento:

```
Bicicletas mas usadas:
id: 705
Bicicletas menos usadas:
id: 734
Recorrido mas realizado:
id origen: 3
id destino: 7
Tiempo promedio de uso: 27,245180
Bicicletas utilizada mas tiempo :
id: 705
Tiempo de la bicicleta mas utilizada : 4164600,000000
Tiempo de ejecucion en segundos: 4,149000
```

Luego de aplicar los cambios procedimos aplicar la solución para las nuevas funcionalidades:

Para agregar el tiempo de la bicicleta más usada y su respectivo id se procedió de la siguiente manera: - Se agrego una clase TimeAndQuantityBike que almacena el tiempo y las veces que fue usada una bicicleta. Cada objeto creado de esta clase fue almacenada como value en el mapa mapBike.

Para calcular el tiempo de procesamiento se agregaron dos líneas de código (this.startTime = System.currentTimeMillis() y long endTime = System.currentTimeMillis() - this.startTime;) en las clases **StatisticalProcessorDeamonStrategy.java** y **StatisticalProcessorOnDemandStrategy.java.**

Para la impresión en el archivo de salida se agregaron las líneas correspondientes en la clase **YamlExporter.java**

Otras correcciones que se efectuaron fueron adaptar la solución a lo pedido por la cátedra para su ejecución y salida.

Antes de las correcciones, el software se ejecutada de la siguiente manera: java aydoo.Processor.MainStatisticalProcessor. <Path>, mientras la forma correcta que fue establecida por la cátedra es la siguiente: java aydoo.Procesador <Path>

La salida de los archivos .yml no se realizaba en una carpeta Salida dentro de Build, por lo que se corrigió.

El software también contaba con un error, al encontrar un registro que no contenga tiempo de uso, lanzaba una exception y dejaba de ejecutar el software. Para solucionar este inconveniente se recurrió a la siguiente solución en la clase ParserZipDeamon.java:

```
String time;
    if (row.length == 8)
        time = "0";
    else
        time = row[8];
```

Otro error que detectamos es que el software no aceptaba los archivos CSV separados por punto y coma, por lo que los archivos de entrada estaban modificados y cambiados por comas en su lugar. Como solución, modificamos en la clase ParserZipDeamon.java, dentro del método proccesTravel(InputStream stream) la línea que decía: `String cvsSplitBy = ";";`

En resumen las clases que se vieron afectadas fueron las siguientes:

**ParserZipDeamon.java**
**ParserZipOnDemand.java**
**YamlExporter.java**
**StatisticalProcessorDeamonStrategy.java**
**StatisticalProcessorOnDemandStrategy.java**
**StatisticalProcessorStrategy.java**
**MainStatisticalProcessor.java Procesador.java**
**TimeAndQuantityBike.java**

Como es requisito de la cátedra que el software corra bajo el sistema linux, este fue probado en sevidores externos (Telecentro) a nuestras computadoras, por lo que vimos que dependiendo del tráfico en el momento de la ejecución del modo Daemon, la transferencia de los archivos zip (14mb) variaba entre 5 y 8 segundos.

Dado que el monitor de la carpeta supervisada de nuestro sistema, tomaba estos archivos antes que se complete la transferencia generando una exception, nos parecio prudente agregar el siguiente código en la clase StatisticalProcessorDeamonStrategy dentro de la captura del evento, para generar un tiempo y dar lugar a la finalización de la trasferencia

```
try {
        Thread.sleep(5000);
} catch (InterruptedException e) {
        e.printStackTrace();
}
```

Lo que aprendimos:

Aprendimos de lo vital que resulta hacer un buen diseño desde el principio para que luego sea más fácil el mantenimiento y la aplicación de nuevas funcionalidades.

Pudimos aplicar en forma práctica atributos de calidad y buenas técnicas de programación.

Finalmente, nos llevamos como muy positivo para nuestra vida profesional el uso de nuevas herramientas, vitales a la hora de diseñar y programar, tales como Travis, Jenkins, Cobertura y Pmd entre otras. Su uso nos ayudó bastante para mejorar respecto a las buenas prácticas de diseño.