Informe Obligatorio Programación II



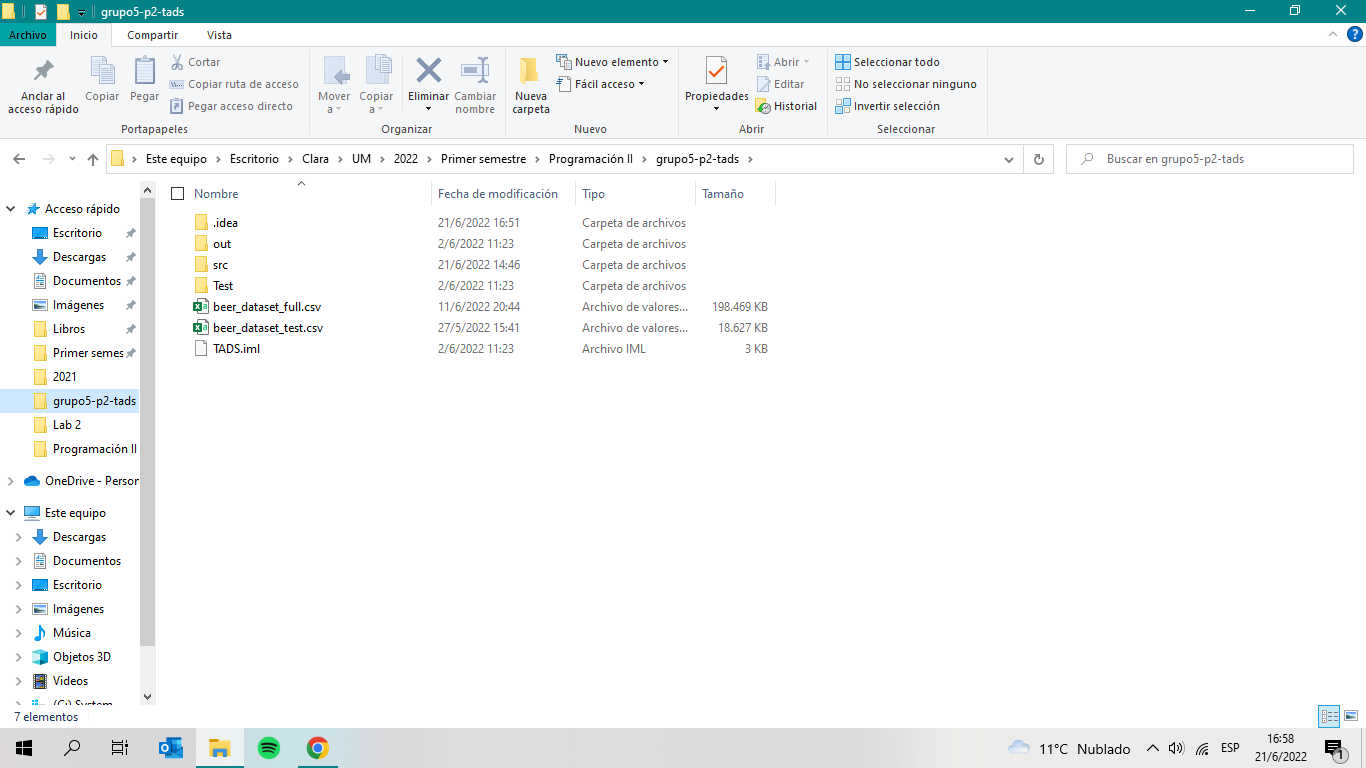
LINK A GITHUB:

<https://github.com/fedevaz0412/grupo5-p2-tads>

ACLARACIÓN:

Para correr el programa se debe tener a los archivos “csv.” en la raíz del proyecto.

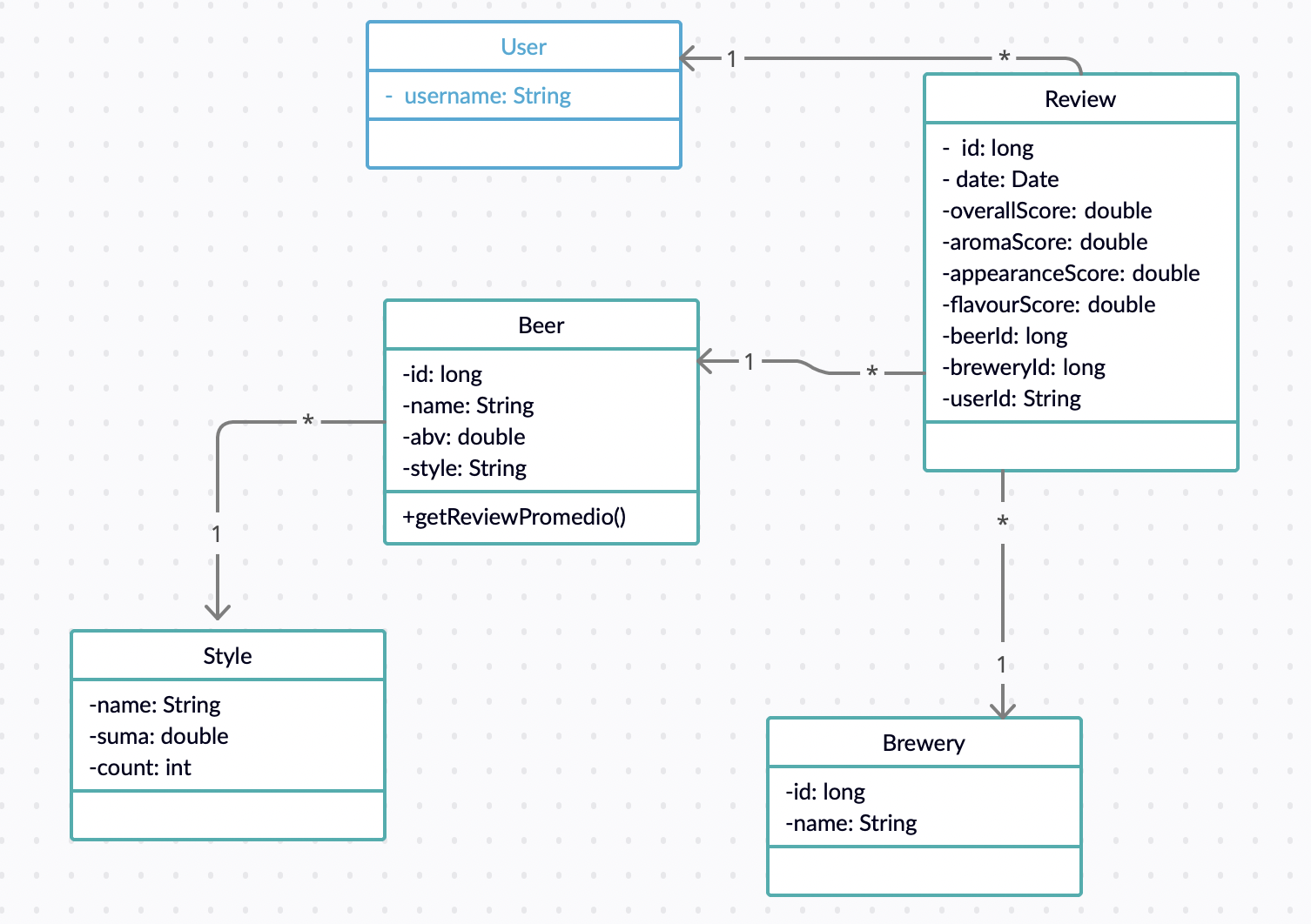
De esta manera:



Clara Echeverría

Federico Vázquez

***Diagrama UML:***



No tuvimos que implementar entidades adicionales. Además de las entidades, tenemos una carpeta Util, donde implementamos la carga de datos, las consultas y conversores (De fecha y numéricos) para guardar correctamente los datos recibidos del csv.

En la clase main está implementado el menú principal con un menú de consultas.

***Carga de datos y reportes:***

Carga de datos:

Para guardar todas las instancias de clases obtenidas desde el csv utilizamos el TAD Hash cerrado, pues nos proporciona gran manejo de cantidades masivas de datos y búsquedas por claves. Cada instancia era individual y contenía un atributo característico que facilitó el acceso a los datos.

La principal ventaja del hash sobre otras estructuras de datos es la velocidad. El tiempo de acceso a un elemento es, en promedio, O(1), por lo que la búsqueda se puede realizar muy rápidamente. Además, el hash es muy eficiente cuando se puede predecir el número máximo de entradas, como era nuestro caso.

A su vez, implementamos en nuestro hash un ArrayList, también creado por nosotros, que contiene todas las keys del hash. Esto lo utilizamos para tener un acceso aún más fácil a las keys.

Para la carga de datos tuvimos en cuenta que si en el archivo .csv alguna de las columnas estaba vacía, entonces esa línea se ignoraba y no se cargaban los datos pertenecientes al resto de las columnas.

Reporte 1:

Se crea un hash cerrado, a partir del hash con todas las reviews, que va a tener todas las reviews que se hicieron en el año ingresado. Esto se hace comparando la fecha de cada review, el año con el año ingresado por el usuario.

Luego creamos otro hash cerrado (hashCons1) con el id de la brewery y la cantidad de reviews para cada una. Obtenemos el breweryId de la review y chequeamos si ya se encuentra en el hash. Si se encuentra, le sumo uno en la cantidad de reviews, si no se encuentra lo agregamos con una review, la primera.

Se crea un heapMax, basado en nuestro hashCons1, con la brewery y como key la cantidad de reviews. De esta forma, usando el delete del heap en un for limitado a 10 (por letra), podemos obtener el resultado ya ordenado.

Reporte 2:

Se crea un array con todas las claves del hash reviews, en este caso idReview. Creamos un hash cerrado (hashCons2) con los usuarios y cantidad de reviews por usuario. Obtenemos el userName del usuario que realizó la review en base al idReview y chequeamos si ya se encuentra en el hash. Si se encuentra, le sumo uno en la cantidad de reviews, si no se encuentra lo agregamos con la primera review.

Se crea otro array con las claves de hashCons2 y un heapMax con la cantidad de reviews y usuarios del tipo User.

Por último, haciendo delete del heap en un for limitado a 15 (por letra), podemos obtener el resultado esperado.

Reporte 3:

Para realizar este reporte es necesario pedir por consola la fecha de inicio y final en que se va a realizar la búsqueda. Validamos que estas fechas sean del tipo Date en la clase Main.

Creamos un array con las claves de las reviews e iteramos en las reviews (utilizando el tipo Review). Con el getter de Date y la función compareTo obtenemos las fechas del rango especificado por consola y devolvemos la cantidad de reviews en el rango.

Reporte 4:

Se crea un array con las claves del hash reviews y un hash cerrado que contendrá el nombre del estilo de cerveza y un objeto Estilo. Mediante la review del array obtenemos la cerveza en cuestión y luego el estilo de la misma. Con un getter en Style encontramos la suma de puntajes anterior (al cargar los datos estos atributos no se consideran, quedan en cero por defecto) y le sumamos la de la review actual. Con otro getter en Style tenemos la cantidad de reviews que se hicieron para este estilo. Con setters se actualizan los valores de las mismas.

Por último creamos un heapMax con el promedio del puntaje y el Style. Con un delete en un for limitado en 7 (por letra), para cada estilo hacemos el promedio del puntaje con la información obtenida anteriormente y devolvemos la información solicitada.

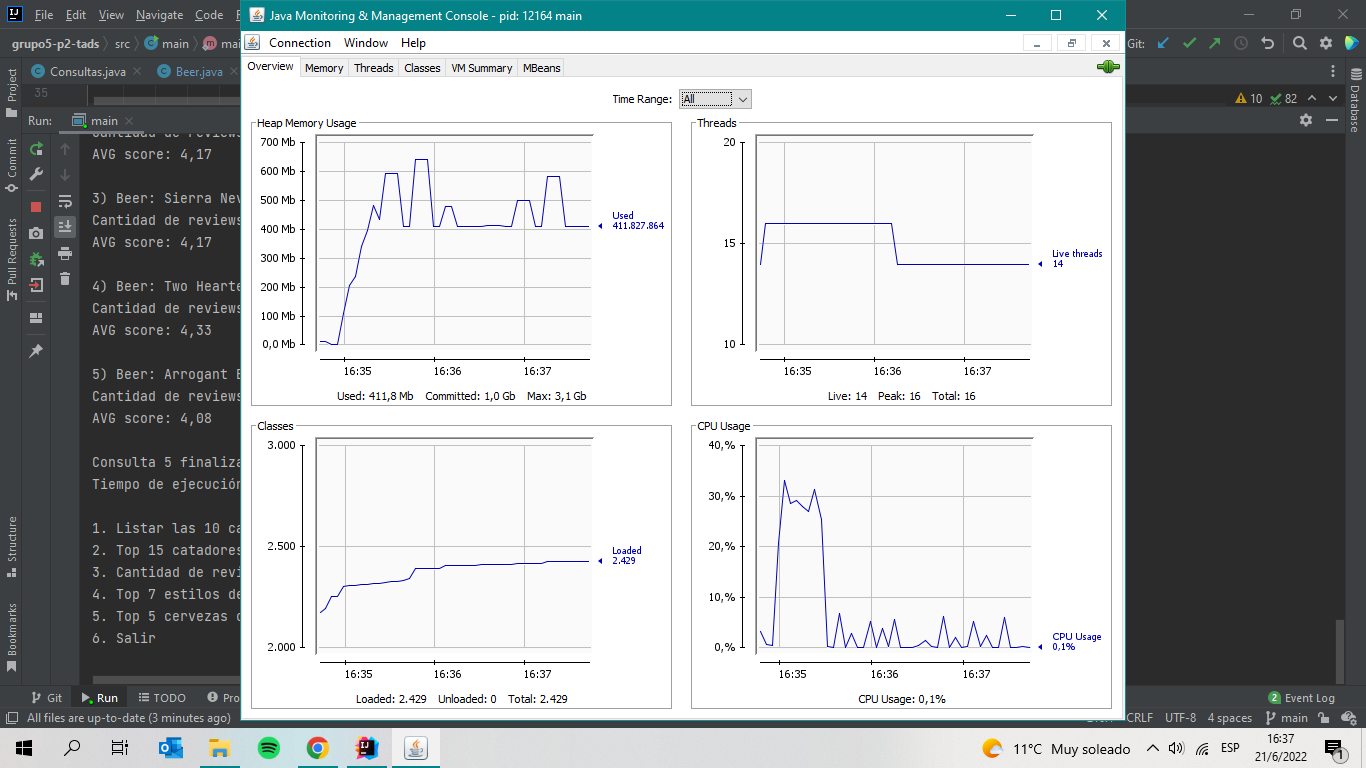
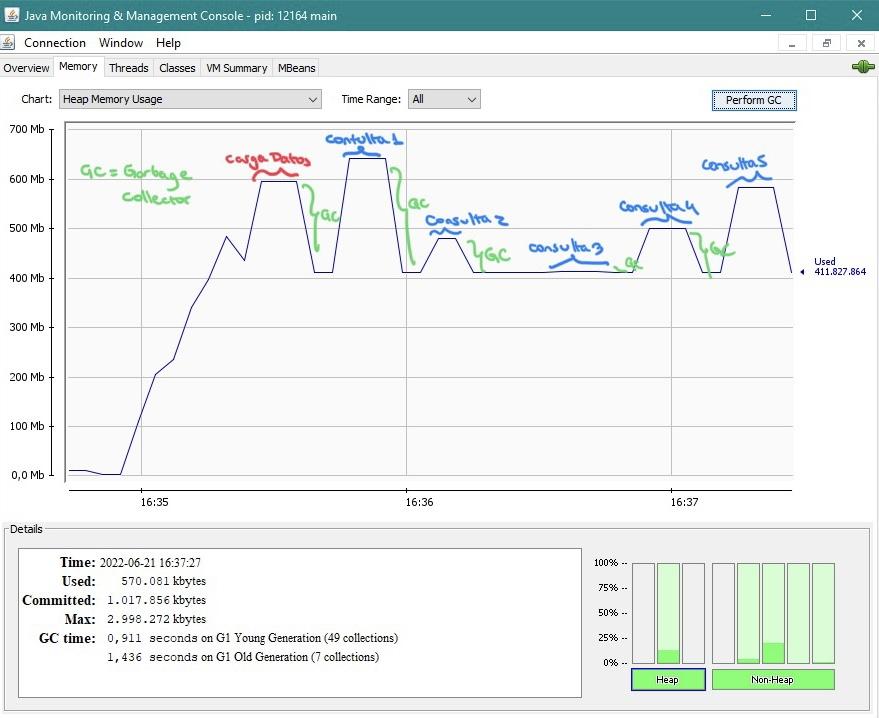
Reporte 5:

Se crea un array con todas las claves del hash reviews, en este caso idReview. Creamos un hash cerrado (hashCons5) con las cervezas y cantidad de reviews por cerveza. Obtenemos el beerId de la cerveza que fue punteada en base al idReview y chequeamos si ya se encuentra en el hash. Si se encuentra, le sumo uno en la cantidad de reviews, si no se encuentra la agregamos con la primera review.

Se crea otro array con las claves de hashCons5 y un heapMax con la cantidad de reviews y cervezas del tipo Beer.

Por último, haciendo delete del heap en un for limitado a 5 (por letra), obtenemos los resultados esperados. Para lograr esto, creamos un método getReviewPromedio en la entidad Beer, que en base a la cantidad total de reviews de una cerveza y la suma de todos los puntajes obtenidos se devuelve el promedio del puntaje.

***Medición de eficiencia de la aplicación:***



***Análisis:***

Al momento de hacer la carga de datos vemos que el uso de memoria sube a 600Mb porque implementa los 5 hash cerrados para guardar los datos desde el csv. Al hacer el garbage collector baja a 400Mb (Cada vez que se utiliza, la memoria baja al mismo valor).

Al correr la primera consulta, el uso de memoria sube a 650Mb, el valor más alto respecto a las otras consultas. Esto se debe a que se implementan 2 hash cerrados y un heap, y en las otras consultas como máximo 1 hash y un heap.

Con respecto a la segunda y cuarta consulta, que usan 1 hash y 1 heap, el uso de memoria es prácticamente el mismo, 500Mb.

En la tercera consulta, el uso de memoria apenas varía con respecto al valor base (el obtenido después de un GC) y es lógico porque no se utiliza ningún hash ni heap.

La quinta consulta usa la misma cantidad de hash y heap que la primera y cuarta, pero además utiliza un método en la clase beer, para obtener el promedio de reviews de una cerveza, por esto es entendible el aumento en el uso de memoria a 580Mb.

En cuanto a los tiempos de ejecución, al correr el main está detallado para cada caso, sea la carga de datos o las diferentes consultas, cuánto demora en hacerlo. Para la carga del archivo beer\_dataset\_full.csv demora entre 25 y 35 segundos y para todas las consultas demora menos de un segundo.