- **Enunciado:**

  Siguiendo el modelo de implementación del concepto de Tabla Hash con Listas de Desborde, se solicita ahora implementar una clase *TSB_OAHashtable* (por *TSB Open Addressing Hash Table*), que represente una *Tabla Hash* pero con estrategia de *Direccionamiento Abierto* para la resolución de colisiones.

  La clase debe ser implementada en forma rigurosa, siguiendo el modelo ya presentado para la clase *TSBHashtable*. Esto implica:

  - Implementar la interface *Map<K, V>* y desde ella, los mismos métodos que se implementaron para la clase *TSBHashtable*.
  - Definir dentro de la clase *TSB_OAHashtable* una clase interna *Entry* que implemente la interface *Map.Entry<K, V>* para representar a cada par que se almacene en la tabla.
  - Definir dentro de la clase *TSB_OAHashtable* las tres clases internas para gestionar las *vistas stateless* de *claves*, de *valores* y *de pares* de la tabla, incluyendo a su vez en ellas las clases internas para representar a los *iteradores* asociados a cada vista.
  - Redefinir en la clase *TSB_OAHashtable* los métodos *equals()*, *hashCode()*, *clone()* y *toString()*que se heredan desde *Object*.
  - Definir en la clase *TSB_OAHashtable* los métodos *rehash()* y *contains(value)* que no vienen especificados por *Map*, pero son especialmente propios de la clase (emulando a *java.util.Hashtable*).

  La idea es tomar como modelo al que se presentó para la clase *TSBHashtable* en la Ficha 10, e implementar los mismos métodos publicos y protegidos de esa clase, más las clases internas citadas. Obviamente, con relación a los atributos privados y métodos privados, cada grupo de trabajo hará su propia propuesta y diseño.

  Luego de implementar la clase pedida, se debe desarrollar un programa que sea capaz de procesar un conjunto de *archivos de texto* y construir una *tabla hash con todas las palabras diferentes* descubiertas en esos archivos, de forma de determinar además la *frecuencia de aparición* de cada palabra en esos archivos. Para ello, el programa debe ser capaz de procesar de a uno los archivos indicados (puede descargarlos desde aquí), obteniendo de cada uno las distintas palabras detectadas. Notar aquí que para obtener las palabras se deberá analizar (o "parsear") cada documento, limpiar el texto de los signos de puntuación, palabras numéricas o alfanuméricas, dígitos y todo otro símbolo que no forme parte de una palabra normalmente entendida; y una vez realizado este trabajo (o a medida que se va realizando) se procese cada palabra agregándola a la tabla y contando su frecuencia de aparición.

  Como se está trabajando con una tabla tipo *Map*, y considerando que luego se pedirá buscar una palabra e indicar cuántas veces apareció, se sugiere que en cada par *(key, value)* que se almacene en la tabla el objeto *key* sea la palabra detectada en el texto, y el objeto *value* sea la frecuencia de aparición de esa palabra.

- **Requerimientos:**

  El programa a desarrollar debe cumplir con los siguientes requerimientos:

  a.) Contar con interfaz de usuario simple, pero basada en ventanas y componentes visuales en base al framework Java FX (que será oportunamente tratado en clases y con ficha de soporte).

  b.) Prestar especial atención en cuanto a factores de eficiencia: pensar en un diseño eficiente para no caer en casos en los que el proceso de los archivos incurra en una cantidad exagerada de tiempo y vuelva nuestro programa una alternativa poco viable de ser utilizada en la realidad.

  c.) Permitir que la tabla hash, una vez generada, sea grabada en un archivo serializado. Y permitir recuperar esa tabla desde el archivo serializado. Esto puede hacerse en forma simple: al terminar de crear la tabla, grabarla por serialización en forma automática y dejarla allí. Y cada vez que el programa sea ejecutado, levantar la tabla también en forma automática desde el archivo serializado (a menos que el archivo no exista).

d.) Mostrar en todo momento la cantidad total de palabras distintas que contiene la tabla. Permitir además, ingresar una palabra cualquiera en un campo de texto, buscar esa palabra en la tabla y mostrar su frecuencia de aparición.

h.) Permitir agregar documentos nuevos. Al presentar el trabajo, la tabla debe estar ya generada y serializada en base a los archivos de texto dados (cada alumno debe entregar el proyecto Java, y el archivo serializado que contiene a la tabla ya creada). Al volver a arrancar el programa, el mismo debe volver a levantar la tabla (como se indicó en el punto c). Pero el programa debe permitir que se procese uno o varios archivos de texto más se actualice la tabla con las nuevas frecuencias o nuevas palabras de acuerdo a el/los nuevos archivos procesados.

- **Entrega:**

  El trabajo es de *realización grupal*. Debe ser realizado en grupos de no menos de 2(dos) personas y no más de 4(cuatro) como máximo.

  Cada integrante del grupo debe subir al aula virtual su copia del trabajo en un archivo comprimido conteniendo el proyecto desarrollado en Java y el archivo serializado con la tabla grabada. Obviamente, todos los integrantes de un mismo grupo deberán subir la misma copia… pero TODOS deben hacerlo.

  El archivo comprimido mencionado debe ser nombrado incluyendo los apellidos de los integrantes del grupo separados por guiones por ejemplo si el práctico lo presentaran los jtp de la cátedra debería llamarse *Serrano-Teicher-Tartabini.zip* (o .rar o 7z o cualquier tipo de compresión que se pueda descomprimir con 7Zip).

- **Consideraciones:**

  Noten los estudiantes que el desarollo del trabajo puede hacerse en forma gradual:

  - Aún si la clase *TSB_OAHashtable* no estuviese terminada de implementar, pueden parsear de todas formas los documentos y almacenar las palabras en una instancia de la clase *TSBHashtable* provista por la Cátedra con la Ficha 10. Dado que ambas clases implementan la interface *Map*, cuando terminen efectivamente de implementar la clase *TSB_OAHashtable*, simplemente cambien el tipo de la instancia creada y el polimorfismo hará el resto sin cambiar nada más.
  - Con respecto a la interfaz de usuario, el programa puede plantearse en base a interfaz de consola estándar mientras los estudiantes se familiarizan con el framework *Java FX*. Y ciuando hayan visto cómo aplicar *Java FX*, migrar la interfaz de usuario de consola estándar a ventanas *Java FX*.