Trabajo TP6 de Fundamentos de Informática

Primer Curso Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2021-2022

Escuela de Ingeniería y Arquitectura

Departamento de Informática e Ingeniería de Sistemas

Área de Lenguajes y Sistemas Informáticos

v 1.0 Octubre 2021

Análisis de redes sociales

Objetivo del trabajo

El objetivo de este trabajo es realizar un programa en el que puedas aplicar todos los conocimientos y habilidades adquiridas durante el curso. En esta ocasión, el dominio del problema es el análisis de los contenidos publicados en la Redes Sociales, concretamente, de los mensajes publicados en Twitter en torno a la erupción del volcán de la Palma.

Descripción del problema

La erupción del volcán de la Palma se ha convertido en un fenómeno ampliamente comentado en las Redes Sociales, tanto por organismos oficiales como usuarios particulares. Twitter no ha sido ajeno a este fenómeno y son numerosos los mensajes que se pueden encontrar en esta red en torno a este tema de actualidad. Dado que Twitter permite descargarse en formato fichero de texto parte de los mensajes que publican sus usuarios, vamos a aprovecharnos de esa funcionalidad para acceder a algunos de esos mensajes y procesar y analizar su contenido.

La estructura del fichero de texto descargado es la siguiente. La primera línea es lo que se denomina la *cabecera del fichero*, es decir, una descripción de cómo se estructuran los datos contenidos a partir de la segunda línea. Después la información de cada *tweet* es almacenada en una línea de texto, conforme a lo descrito en la cabecera, y debería contener los siguientes campos: la fecha/hora en la que se publicó el mensaje, el tipo de aplicación que utilizó el usuario para publicar el mensaje, el identificador del publicador y, finalmente, el contenido del mensaje. Estos campos están separados entre sí por un carácter punto y coma.

Si analizamos visualmente el contenido de los mensajes (último campo de cada tweet), podemos observar que es frecuente que contenga "*tags*" (los cuales comienzan con el carácter '#'), *URL*s (en formato Web http://...), emoticonos, etc. También se puede observar que contiene diferentes signos de puntuación (comas, puntos, dos puntos, punto y coma, etc.). Todos estos elementos deberán ser tratados adecuadamente como parte del trabajo de procesado del fichero, conforme a lo que se explica en el enunciado de las distintas actividades.

El objetivo del trabajo es procesar los tweets del fichero proporcionado para extraer cierto conocimiento de interés.

Notas de interés sobre la realización del trabajo

Es importante que antes de comenzar a trabajar leas con detenimiento las siguientes notas:

- **El trabajo es individual**. Las copias serán calificadas con un cero (a todos los estudiantes implicados, sin excepción).
- **El trabajo constará de tres partes bien diferenciadas**, y cada parte de una o varias tareas sencillas.
- **Para superar el trabajo TP6 será necesario realizar y entregar en plazos las dos primeras partes, pudiendo obtener una nota máxima de 7.0. La tercera parte será "opcional" y permitirá aspirar a la calificación máxima, un 10.0.**
- **La naturaleza del trabajo es no presencial**, aunque los estudiantes dispondrán de sesiones de tutoría concretas para esta actividad.

Notas de interés sobre la entrega final del trabajo

Es importante que desde el primer día tengas presente en todo momento las siguientes notas:

- **Fechas de entrega:** El trabajo TP6 tiene como fecha límite de entrega el miércoles 9/1/2022, a las 23:59h, horario peninsular. Esta fecha es no negociable.
- **Debes entregar** en un único fichero comprimido, llamado `tweetsLP_NIP.zip` (donde NIP debe sustituirse por el NIP del estudiante), todos los ficheros fuente que has programado (sólo los ficheros fuente .java). No puedes utilizar ninguna librería que no haya sido utilizada, explicada o comentada durante el curso o en este guion.
- **La evaluación del trabajo será presencial y tendrá lugar los días 13 y 14 de enero del 2022, aunque estas fechas son aún tentativas y están pendientes de confirmar**. La fecha/hora y lugar en la que cada estudiante tendrá que exponer y defender su trabajo será anunciada con suficiente antelación a través de la plataforma Moodle. Sólo podrán defender su trabajo aquellos estudiantes que lo hayan entregado antes de la fecha límite establecida. La no asistencia a la defensa implica una calificación de No Presentado.

Notas relativas a la gestión del trabajo

Cada estudiante deberá llevar un control individual de la dedicación en horas que invierte en el trabajo de la asignatura. Como parte del material entregado, se ha publicado un documento Excel "*controlDedicacion.xls*" donde deberéis apuntar estas horas. La unidad de control son las horas, por ejemplo, si un día dedicáis 1 hora y 12 minutos, tendréis que anotar 1,25 horas: o si dedicáis 38 minutos, apuntaréis 0,75 horas.

El control de dedicación es obligatorio. **El profesor lo podrá solicitar en cualquier instante para comprobar que se está realizando correctamente**, penalizando si éste no se está completando.

Notas de interés sobre las tutorías del TP6

Los estudiantes dispondrán de 2,5h a la semana de **tutorías especializadas** (**tutorías TP6**) en esta actividad, conforme a los siguientes horarios:

- **Grupo de mañana:** los lunes lectivos de 15:30h a 18h.
- **Grupo de tardes:** los viernes lectivos de 9:30h a 12h.

Es necesario **reservar un turno de "Tutoría TP6" con SUFICIENTE ANTELACIÓN** a través del Calendario Google de tutorías de Pedro Álvarez (la información está en Moodle). Podrás comprobar qué **turnos de "tutorías TP6"** están libres y reservar el que más te interese (uno por día). Al hacer la reserva se te enviará una notificación que contiene un enlace a ***Google Meet***. Dependiendo de la situación sanitaria, estas tutorías se realizarán de forma presencial o telemática a través del enlace generado.

PRIMERA PARTE DEL TRABAJO

Cuando se quieren aplicar técnicas de análisis sobre ficheros de datos es preciso siempre realizar ciertos procesamientos y filtrados previos. El objetivo es "limpiar" de errores e información innecesaria el fichero original, quedándote únicamente con la parte del fichero que te interesa. Estas tareas de pre-procesamiento suelen ser costosas, pero son imprescindibles.

A pesar del cuidado puesto en el proceso de generación del fichero de tweets "*tweets-LaPalma.txt*", éste aún contiene algunos errores y deficiencias que deben ser resueltos. Por este motivo, el objetivo de esta primera parte del trabajo TP6 es que completéis este pre-procesamiento realizando las tareas que se proponen a continuación.

Tarea 1. Pre-procesamiento del fichero para eliminar los caracteres extraños.

Programar un método que pre-procese un fichero de entrada de tweets (el nombre del fichero se especifica a través del parámetro "*input*") con objeto de sustituir por espacios en blanco todos aquellos caracteres no deseables. El resultado de este proceso de filtrado se almacenará en un fichero de salida especificado por medio del parámetro "*output*" (nombre del fichero). Se consideran como caracteres válidos: las letras minúsculas y mayúsculas, los dígitos, el punto, la coma, el punto y coma, los dos puntos, los espacios en blanco y los caracteres '_', '-', '#', '@', y '/'. Todos los demás caracteres deberán ser sustituidos, cada carácter por un espacio en blanco. Además, el fichero de salida deberá contener la cabecera del fichero original y respetar su estructura de líneas.

Adicionalmente, el método programado escribirá por pantalla el número de caracteres sustituidos, conforme al siguiente formato de ejemplo:

```
--------------------------------------
(Tarea 1) Estadísticas del filtro por contenido:
Número de caracteres sustituidos = 43
--------------------------------------
```

Especificación del método a implementar

```
/**
* filtroContenido: elimina del fichero los caracteres no deseables
* @param input   Nombre del fichero de entrada a procesar
* @param output  Nombre del fichero resultado
*/
public static void filtroContenido(String input, String output){…}
```

Tarea 2. Pre-procesamiento del fichero para eliminar los tweets con formato incorrecto.

Programar un método que pre-procese un fichero de entrada de tweets (el nombre del fichero se especifica a través del parámetro "*input*") con objeto de eliminar aquellos

mensajes (líneas de fichero) que no tienen la estructura de información adecuada. Un tweet correcto consta de cuatro campos de información separados entre sí por el carácter ';'. El resultado de este proceso de filtrado se almacenará en un fichero de salida especificado por medio del parámetro "*output*" (nombre del fichero). Además, el fichero de salida deberá contener la cabecera del fichero original y respetar su estructura de líneas.

El método también deberá escribir por pantalla el número de tweets que han sido eliminados, conforme al siguiente formato de ejemplo:

```
-------------------------------------
(Tarea 2) Estadísticas del filtro por estructura:
Número de tweets incorrectos eliminados = 12
-------------------------------------
```

Especificación del método a implementar

```
/**
* filtroEstructura: elimina del fichero los tweets que no constan de 4 campos
* de información
* @param name Nombre del fichero de entrada a procesar
* @param res  Nombre de fichero de resultado
*/
public static void filtroEstructura(String input, String output){…}
```

Tarea 3. Pre-procesamiento del fichero para eliminar los espacios en blanco innecesarios.

Como parte de la Tarea 1 se han sustituido un número significativo de caracteres no deseables por espacios en blanco. Esa sustitución provoca que en el contenido de los mensajes (cuarto campo de un tweet) aparezcan secuencias de dos o más caracteres en blanco consecutivos. El objetivo de esta tarea es compactar esas secuencias, sustituyéndolas por un único espacio en blanco.

Programar un método que pre-procese un fichero de entrada de tweets (el nombre del fichero se especifica a través del parámetro "*input*") con objeto de sustituir las secuencias de dos o más espacios en blanco por un único espacio. Este procesamiento sólo se aplica al contenido (o mensaje) de los tweets, es decir, al cuarto campo de información de cada línea. El resultado de este proceso de filtrado se almacenará en un fichero de salida especificado por medio del parámetro "*output*" (nombre del fichero). Además, el fichero de salida deberá contener la cabecera del fichero original y respetar su estructura de líneas.

El método también deberá escribir por pantalla el número de secuencias compactadas, conforme al siguiente formato de ejemplo:

```
-------------------------------------
(Tarea 3) Estadísticas del filtro de secuencias:
Número de secuencias compactadas = 96
-------------------------------------
```

Especificación del método a implementar

```
/**
* compactaBlancos: sustituye dos o más caracteres ' ' consecutivos por una
* sola ocurrencia
* @param input   Nombre del fichero de entrada a procesar
* @param output  Nombre del fichero de resultado
*/
public static void compactaBlancos(String input, String output) {…}
```

Tarea 4. Extracción de los "tags" utilizados por los usuarios.

El objetivo de esta tarea es extraer la lista de "tags" utilizados por los usuarios en los tweets que publican (no debe contener "tags" repetidos). Un "tag" está formado por el carácter '#' y una secuencia de uno o más caracteres, por ejemplo, "#LaPalma" o "#Volcan2021_LaPalma". El proceso de reconocimiento de "tags" no es sensitivo a letras minúsculas y mayúsculas, es decir, consideraremos que "#LaPalma" y "#lapalma" son el mismo "tag".

Antes de programar la extracción de "tags" es necesario implementar los siguientes métodos:

Cálculo del número de "tags" contenidos en una línea de texto

Dada una línea de texto, especificada como parámetro de entrada ("*línea*"), el método devuelve como resultado el número total de "tags" encontrados.

Especificación del método a implementar

```
/**
* numTagLinea: devuelve el número de tags encontrados en la línea de texto
* de entrada
* @param linea  Línea de texto a procesar
* @return       El número de tags encontrados
*/
public static int numTagLinea(String linea) {…}
```

Extracción de los "tags" contenidos en una línea de texto

Dada una línea de texto, especificada como parámetro de entrada ("*línea*"), el método devuelve los "tag" encontrados en dicha línea. El resultado es un vector de *String*, donde cada componente almacenará un "tag" concreto. Este vector puede contener "tags" repetidos. Si la línea no contiene ningún "tag", el resultado debe ser *null*.

Especificación del método a implementar

```
/**
* extraccionTagLinea: devuelve los tag (como String) encontrados en una
* línea de texto de entrada
* @param linea  Línea de texto a procesar
* @return       Los tags encontrados.
*               Si no encuentra ninguno devuelve el valor null
```

```
*/
public static String[] extraccionTagLinea(String linea) {…}
```

Chequea si existe un "tag" en un conjunto

En Java una estructura de tipo *ArrayList<String>* es similar a un vector de datos *String*. En el **Anexo 1** de este enunciado se explica en detalle este tipo de estructura y la forma de utilizarla. El objetivo de este método es, dado un conjunto de "tags" almacenados en una estructura *ArrayList<String>* (primer parámetro), comprueba si un "tag" concreto (especificado como segundo parámetro) está contenido en la estructura. El resultado del método es un booleano que representa si está (true) o no contenido (false).

Especificación del método a implementar

```
/**
* existeTag: chequea si un tag está contenido en un conjunto de tags
* @param tags   Estructura que contiene el conjunto de tags
* @param tag    Tag a buscar
* @return       Si el tag especificado está contenido en el conjunto
*/
public static boolean existeTag(ArrayList<String> tags, String tag) {…}
```

Una vez programados estos métodos, el objetivo final de la tarea es programar el método de extracción a nivel de ficheros.

Especificación del método a implementar

```
/**
* extraccionTagsFichero: devuelve todos los tags diferentes encontrados
* en un fichero de texto de entrada
* @param input   Nombre del fichero a procesar
* @return        Estructura de datos que contiene los tags
*/
public static ArrayList<String> extraccionTagsFichero(String input) {…}
```

El método debe procesar un fichero de entrada de tweets (el nombre de este fichero se especifica a través del parámetro "input"), extraer todos los "tags" diferentes que en él están contenidos y devolverlos como resultado en una estructura de tipo *ArrayList<String>*.

El método también deberá escribir por pantalla el número de "tags" encontrados, conforme al siguiente formato de ejemplo:

```
-------------------------------------
(Tarea 4) Estadísticas de la extracción de tags:
Número de tags diferentes encontrados = 43
-------------------------------------
```

Tarea 5. Primera versión del programa principal

Como parte del material de este trabajo se proporciona una primera versión del programa principal del sistema de análisis de tweets. Esta versión invoca a los métodos anteriores en el orden correcto. El objetivo de esta tarea es integrar los métodos anteriores en este programa principal y comprobar que funcionan correctamente.

¿Cómo realizar esta comprobación? Se proporciona un fichero de texto, llamado "*tweets-de-prueba.txt*", con sólo 10 tweets. Por tanto, este fichero es muy simple con respecto a los ficheros que el programa debería procesar. Tenéis que usar este fichero para chequear que vuestro programa funciona. Se recomienda que abráis los distintos ficheros temporales que se generan como resultado de los distintos métodos y visualmente comprobéis que estáis resolviendo cada tarea concreta. También podéis buscar visualmente qué "tags" deberíais extraer y comprobar si corresponden con vuestra solución.

Una vez os aseguréis que el programa funciona, lo debéis probar un fichero de datos real. Como parte del material se proporciona un fichero de datos más complejo que el anterior de prueba, llamado "*tweets-LaPalma.txt*".

ANEXO 1: Estructuras de tipo ArrayList<>

Este anexo tiene como objetivo explicar brevemente cómo trabajar con una estructura *ArrayList* de Java. Estas estructuras son similares a los *Array* presentados en clase, donde la diferencia principal es que no es necesario fijar su tamaño o número de componentes en el momento de la declaración (se trata de una estructura de naturaleza dinámica). En el trabajo nos limitaremos a utilizar *ArrayList* de cadenas.

- Declaración de un *ArrayList* de cadenas:

```
ArrayList<String> students = new ArrayList<String>();
```

Nótese que no se especifica un tamaño concreto para la estructura.

- Operaciones necesarias para manipular un *ArrayList* de cadenas:

```
String ejemplo = new String();

students.add(ejemplo);    // Almacena la cadena ejemplo en la estructura

boolean existe = students.contains(ejemplo);  // Comprueba si existe un String
ejemplo en la estructura y guarda el resultado en el booleano existe

int numDatos = std.size();  // Almacena en numDatos el número de datos String
almacenados en ese instante en la estructura

String cadena = std.get(2);  // Accede al String almacenado en la componente 2
de la estructura y lo guarda en la variable String cadena
```

En la siguiente página Web podéis acceder a más información sobre el uso y las operaciones disponibles en torno a las estructuras *ArrayList*:

https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

SEGUNDA PARTE DEL TRABAJO

La segunda parte del trabajo no presencial consta de las siguientes tareas.

Tarea 6. Obtener los "tag" más utilizados

En la tarea 4 extrajisteis los distintos "tags" utilizados por los usuarios. El objetivo de esta tarea es contar cuántas veces aparece cada "tag" en los "tweets" que están siendo procesados e informar de cuáles son los "tag" más populares, indicando para cada uno de ellos el número de ocurrencias encontradas.

Para completar esta tarea es necesario en primer lugar declararse un nuevo tipo de datos `Tag`, de tipo registro, cuya especificación es la siguiente:

```
public class Tag {

      String texto;
      int ocurrencias;
}
```

Cada instancia es un registro que almacena en el campo `texto` la cadena que representa el "tag" (por ejemplo, "#LaPalma") y en el campo `ocurrencias` el número de veces que ha sido utilizado. Es importante darse cuenta que en la instancia correspondiente a un "tag" estarán contabilizadas todas sus ocurrencias, con independencia de mayúsculas/minúsculas. Es decir, en la instancia del "tag" "#LaPalma" se contabilizarán las apariciones de "#LAPALMA", "#lapalma", "LAPalma", etc. En base a esta especificación, se pide completar las siguientes tareas.

Obtener las ocurrencias de los distintos "tags"

El método recibe como parámetros el fichero de "tweets" a procesar y el conjunto de "tags" diferentes que existen en ese fichero (este conjunto se determina utilizando el método de la tarea 4). El objetivo es contabilizar el número de ocurrencias de cada uno de esos "tag" en el fichero, y devolver el resultado como un vector de registros que contenga la información computada.

Especificación del método a implementar

```
/**
* Devuelve un vector de Tag's, donde cada registro contiene un "tag"
* diferente y el número de ocurrencias de éste en el fichero de entrada
*
* @param input  Nombre del fichero de tweets filtrados
* @param tags   Listado de tags diferentes encontrados en el fichero
* @return Tag[]  Vector de "tags" y sus ocurrencias
*/
public static Tag[] ocurrenciaTagsFichero(String input, ArrayList<String> tags)
```

Ordenar los "tags" en función del número de ocurrencias

Implementar un método que ordene un vector de entrada de datos de tipo `Tag` (parámetro "inputData") según el número de ocurrencias, concretamente, de mayor a menor ocurrencia. Si dos "tag" tienen el mismo número de ocurrencias, en ese caso se ordenan alfabéticamente en base al texto del "tag". La ordenación deberá estar basada en el "*algoritmo de la burbuja*". Si no conocéis el algoritmo, deberéis documentaros a través de Internet. Además, se deberá especificar en un comentario la fuente de dónde se obtuvo la información. Estas cuestiones se revisarán durante la evaluación.

Especificación del método a implementar

```
/**
* Ordena los "tag" del vector de entrada en función del número de
* ocurrencias, de mayor a menor ocurrencias
*
* @param inputData  Vector de "tags" no ordenado
* @return Song[]    Vector de "tags" ordenado
*/
public static Tag[] ordernarTagsPorOcurrencias(Tag[] inputData)
```

Escribir por pantalla los "tag" más populares

El objetivo del método es escribir por pantalla los "tags" más utilizados. Los parámetros de entrada son un vector de "tags" ordenado por número de ocurrencias (de mayor a menor) y un entero que determina el número de "tags" a escribir. Cada "tags" se escribirá en una línea, especificando de forma clara su texto y número de ocurrencias.

Especificación del método a implementar

```
/**
* Escribe por pantalla los "n" tag más usados
*
* @param inputData  Vector de "tags" ordenado
* @param n          Número de "tags" a escribir por pantalla
*/
public static void escribeTagPopulares(Tag[] inputData, int n)
```

Tarea 7. Extracción de las URL referenciadas por un usuario concreto

El objetivo de esta tarea es programar un método que procese un fichero de "tweets" especificado como parámetro de entrada y extraiga todas las direcciones URL diferentes publicadas en los "tweets" (en los formatos "https://..." o "http://...") de un usuario concreto (parámetro "userName"). Las direcciones encontradas serán devueltas en una estructura de tipo "ArrayList<String>".

Además, el método deberá escribir por pantalla, una vez completado el procesado, el nombre del usuario, el número de "tweets" que ha publicado, cuántas direcciones

diferentes se han encontrado en el fichero de entrada y cuántas direcciones estaban repetidas y fueron descartadas, conforme al siguiente formato de ejemplo:

```
--------------------------------------
(Tarea 7) Estadísticas del procesado de direcciones Web de un usuario:
Nombre del usuario: LuisFerrer
Número de tweets publicados por el usuario = 16
Número de direcciones diferentes encontradas  = 5
Número de direcciones repetidas = 3
--------------------------------------
```

Especificación del método a implementar

```
/**
* URLsUsuario: devuelve todas las URL diferentes encontradas en un fichero de
* texto de entrada para un usuario concreto
*
* @param   input           Nombre del fichero a procesar
* @param   userName        Nombre del usuario
* @return  ArrayList<String>  URLs encontradas en el fichero
*/
public static ArrayList<String> URLsUsuario(String input, String userName)
```

Tarea 8. Extracción de las URL referenciadas en un rango de meses concreto

El objetivo de esta tarea es programar un método que procese un fichero de "tweets" especificado como parámetro de entrada y extraiga todas las direcciones URL diferentes contenidas en los "tweets" publicados en los meses comprendidos entre "inicio" y "fin". Se puede asumir que los meses de entrada son valores correctos, comprendidos entre el 1 y el 12. Por simplicidad, se ignora el año en el que fueron publicados los mensajes. Si se especifica el par (2-4) habrá que incluir todas aquellas direcciones diferentes contenidas en mensajes publicados en los meses de febrero, marzo y abril (de cualquier año); si se especifica (10-2), habrá que considerar los meses de noviembre, diciembre, enero y febrero.

```
--------------------------------------
(Tarea 8) Estadísticas del procesado de direcciones Web por meses:
Mes inicial y final: 10-3
Número de direcciones diferentes encontradas  = 19
Número de direcciones repetidas = 55
--------------------------------------
```

Especificación del método a implementar

```
/**
* URLsMeses: devuelve todas las URL diferentes encontradas en un fichero de
* texto de entrada que fueron publicadas en los meses [inicio…final]
*
* @param   input           Nombre del fichero a procesar
* @param   inico           Mes (numérico) de inicio
* @param   final           Mes (numérico) de fin
* @return  ArrayList<String>  URLs encontradas en el fichero
*/
public static ArrayList<String> URLsMeses(String input, int inicio, int final)
```

Tarea 9. Abrir una URL en el navegador

Este método abre una ventana del navegador por defecto instalado en el ordenador y muestra en ella la página Web correspondiente a la dirección URL especificada como parámetro de entrada ("direccionWeb").

Algunas cuestiones importantes para resolver este método: 1) Los métodos de las tareas 7 y 8 os devuelven algunas de las direcciones URL encontradas y que podéis utilizar para probar que éste funciona correctamente; y 2) Es necesario que investiguéis por Internet cómo se puede mostrar una página Web desde un programa Java (se recomienda que la solución esté basada en el uso de las clases *URL* y *Desktop* de dicho lenguaje de programación). Para la programación de este método es probable que reutilicéis ideas y/o fragmentos de código disponibles en Internet. Si lo hacéis, deberéis entender el comportamiento de ese código y los efectos que su ejecución produce. Además, se deberá especificar en un comentario la fuente de dónde se obtuvo la información o el código reutilizado. Estas cuestiones se revisarán durante la evaluación.

Especificación del método a implementar

```
/**
* abrirEnlaceWeb: Abre una página Web en el navegador
*
* @param direccionWeb  URL de la página a abrir en el navegador
*/
public static void abrirEnlaceWeb(String direccionWeb)
```

Tarea 10. Segunda versión del programa principal

Se pide programar una segunda versión del programa principal que sirva para comprobar que todos los métodos implementados funcionan correctamente y conforme a su especificación.

TERCERA PARTE DEL TRABAJO

La tercera parte del trabajo no presencial consta de las siguientes tareas.

Tarea 11. Análisis mensual de "tag"

Se dispone de un conjunto de ficheros de texto que almacenan colecciones de tweets, conforme el formato del enunciado (el parámetro *files* almacena los nombres de estos ficheros). El objetivo de este método es procesar estos ficheros para determinar qué cinco "tag" han sido los más utilizados en un mes concreto (parámetro *month*). Como resultado de este procesamiento, el método deberá escribir por pantalla la siguiente información:

- Para cada uno de los ficheros de entrada, el nombre del fichero y el número de tweets publicados en el mes *month* que contiene,
- Los cinco "tag" más populares y el número de ocurrencias de cada uno de ellos, considerando todos los ficheros en conjunto.

La solución deberá programarse conforme a las siguientes restricciones:

- Los ficheros de entrada se podrán leer una única vez
- Se deben reutilizar los métodos programados en tareas anteriores

Especificación del método a implementar

```
/**
* Escribe por pantalla los "tag" más utilizados en un mes concreto a partir
* la información almacenada en una colección de ficheros de tweets
*
* @param files  Vector de nombres de fichero (tweets)
* @param month  Mes de referencia
*/
public static void analisisTagMensual(String[] files, int month)
```

Tarea 12. Histograma mensual de "tag"

Se dispone de un conjunto de ficheros de texto que almacenan colecciones de tweets, conforme el formato del enunciado (el parámetro *files* almacena los nombres de estos ficheros). El objetivo el método es contar cuántos tweets por mes contienen un determinado "tag" (parámetro *tag*), considerando globalmente los ficheros, y representar esa información en pantalla en forma de histograma. Por ejemplo,

Tag: #LaPalma
*Enero (12): ****
*Febrero (7): ***
*Marzo (32): ********
Abril (0):
…
*Diciembre (14): ****

representa un ejemplo de histograma concreto, donde el "tag" buscado es "*#LaPalma*", en el que se han encontrado 12 ocurrencia publicadas en enero, 7 en febrero, 32 en marzo, etc. El resultado del método debe respetar este formato de representación: una línea por mes, indicado el número total de ocurrencias y representando ese número por medio de asteriscos. Nótese que se debe usar un asterisco para representar cada 5 (o menos) ocurrencias. Por ejemplo, en el caso de las 12 ocurrencias de enero se utilizan 3 asteriscos.

La solución deberá programarse conforme a las siguientes restricciones y recomendaciones:

- Los ficheros de entrada se podrán leer una única vez
- Se deben reutilizar los métodos programados en tareas anteriores
- Se recomienda diseñar nuevos métodos auxiliares que simplifiquen la solución final

```
/**
* Representa gráficamente en pantalla un histograma de "tag" a partir
* la información almacenada en una colección de ficheros de tweets
*
* @param files  Vector de nombres de fichero (tweets)
* @param tag    Tag de referencia
*/
public static void histogramaTag(String[] files, String tag)
```

Tarea 13. Tercera versión del programa principal

Se pide programar una tercera versión del programa principal que sirva para comprobar que todos los métodos implementados funcionan correctamente y conforme a su especificación.

Nota importante: Una forma sencilla para disponer de un conjunto de ficheros de tweets (tareas 11 y 12) es dividir manualmente un fichero de los disponibles (por ejemplo, el que contiene 2.000 tweets) en varios ficheros (no olvidéis añadir la cabecera en cada uno de los nuevos ficheros).