I.E.S Aguadulce

Titulación: Desarrollo de Aplicaciones Multiplataforma

Asignatura: Programación de servicios y procesos

Tarea número 5

– *Técnicas de programación segura* –

Autor:	David Jiménez Riscardo
Teléfono:	618882196
E-Mail:	david.jimenez.riscardo@gmail.com

Dos Hermanas, 21 de Marzo del año 2022

Table of Contents

En esta tarea trabajaremos con algunos de los conceptos relacionados con la **programación segura** que has visto en la unidad. Para ello tendrás que ir implementando cada uno de los programas que se proponen en los siguientes ejercicios.

Para realizar estos programas **no tienes que desarrollar ningún entorno gráfico**. Serán aplicaciones que se ejecutarán en consola y que pueden recibir parámetros al ser ejecutadas.

1 Filtrado de cuentas válidas

Debes implementar una aplicación Java llamada **ValidadorUsuarios** que reciba como parámetro el nombre de un archivo de texto que contendrá una copia del contenido del archivo /etc/passwd de Linux, en el que se muestra información de las cuentas de usuario registrado en dicho sistema operativo. Cada línea del archivo se corresponderá con una cuenta y tendrá el siguiente formato:

```
<usuario>:<contraseña>:<id_usuario>:<id_grupo>:<descripción_usuario>:<directorio_inicio>:<cascarón>
```

Donde:

1. `<usuario>` será el **nombre del usuario** para la cuenta. Ha de cumplir las siguientes condiciones
   - el tamaño no puede ser **ni menor de seis ni mayor de diez** caracteres;
   - tan solo puede estar formado por **letras minúsculas** y **caracteres numéricos**;
2. `<password>` siempre debe tener el valor x. El carácter representa que la contraseña cifrada está disponible en el archivo */etc/shadow*. El archivo de contraseña no incluye la contraseña por razones de seguridad (todos pueden leerla).
3. `<id_usuario>` Cada usuario creado en una máquina Linux tiene una identificación de usuario única que los identifica en el sistema. Siempre se hace referencia al usuario raíz mediante el ID de usuario 0. Los UID 1 a 99 están reservados para otras cuentas predefinidas, mientras que el sistema reserva los UID 100-999 para cuentas / grupos administrativos y del sistema. Los UID para nuevos usuarios en algunos sistemas Linux comienzan en 1000. Por tanto, este valor debes estar comprendido entre 0 y 2000.
4. `<id_grupo>` Representa el ID único proporcionado para el grupo principal al que pertenece el usuario. Un solo usuario puede ser miembro de varios grupos que se pueden encontrar en el archivo */etc/group*, pero el archivo passwd solo contendrá la información del grupo principal.

5. `<descripción_usuario>` Le permite agregar información adicional sobre los usuarios, como el nombre completo del usuario, el número de teléfono, las descripciones del servicio para el que se creó la cuenta, etc. o incluso, dejar este campo vacío.

6. `<directorio_inicio>` representa la ruta absoluta al directorio de inicio del usuario cuando inicia sesión. Para los usuarios habituales, normalmente sería */home/username,* salvo para *root*, cuyo directorio de inicio es */root*. Por tanto, **este campo debe ser validado solo para usuarios habituales y para root (no para otros usuarios, salvo que tenga el formato /campo1/campo2/campo3...).**

7. `<cascarón>` contiene información sobre el shell predeterminado del usuario. Debemos comprobar que su formato sea /campo1/campo2/campo3....

NOTA: los ángulos simplemente sirven para indicar cada elemento. No hay que escribir nada entre ángulos ni la información viene encerrada entre ángulos.

Algunos ejemplos de líneas podrían ser:

```
franja:x:1020:1021:descripcion:/home/franja:/bin/bash
amalia:x:1002:1002:usuario amalia:/home/amalia:/bin/bash
david:12345:520:520::/home/david:/bin/bash
```

El programa recibirá como argumento desde la consola el nombre del archivo que se desea procesar y llevará a cabo las siguientes acciones:

1. intentará abrir el archivo para su lectura. Si esto no es posible, el programa finalizará ordenadamente y mostrará algún tipo de mensaje de error apropiado;
2. si el archivo se puede abrir, se irá analizando cada línea para comprobar que se trata de una cuenta válida y en caso afirmativo (como es el caso de la primera y la segunda línea del ejemplo) esa información se irá almacenando en memoria. Si una línea no contiene una representación de una cuenta válida (como es el caso de la tercera línea del ejemplo), simplemente se descartará y no se tendrá en cuenta;
3. si se diera el caso de líneas que repiten nombres de usuario, se utilizará siempre el último valor válido que se lea del archivo, descartándose valores anteriores para ese usuario;
4. una vez recorrido el archivo completamente, se mostrará una lista con las cuentas válidas que han sido leídas, ordenadas alfabéticamente por el nombre de usuario.

El **formato de salida final** de las cuentas válidas por pantalla será el siguiente:

```
LISTA DE USUARIOS CORRECTOS
---------------------------
Total de usuarios correctos: <total>
 1: usuario= <xxx_1>     id_usuario= <yyy_1>     id_grupo=<zzz_1>
directorio_inicio=<ttt_1>
 2: usuario= <xxx_2>     id_usuario= <yyy_2>     id_grupo=<zzz_2>
directorio_inicio=<ttt_2>
 3: usuario= ...
 . . .
 . . .
```

donde <total> será la cantidad total de cuentas válidas cargadas, y teniendo en cuenta que estarán ordenadas alfabéticamente por nombres de usuario.

Ejemplo de ejecución

Dado un archivo de texto con el siguiente contenido:

```
root:x:0:0:root:/root:/bin/bash
amalia:Usuario erróneo

linoadmin:x:1000:1000::/home/linoadmin:/bin/bash
franja:x:1001:1001:elprofe:/home/franja:/bin/bash
franja:x:1000000000020:1021::/home/franja:/bin/bash
amalia:x:1002:1002::/home/amalia:/bin/bash
david:x:520:520::/home/david:/bin/bash

linoadmin:x:1000:1000::/home/linoadmin:\bin@bash

tom:x:1000:1000:Vivek Gite:/home/otroTom:/bin/bash

Esta línea es errónea

usuario1:x:500:501:usuario pepito:/home/usuario1:/bin/bash
pepe:12345:1002:1002:Pepe Pótamo,123,981234321,:/home/pepe:/bin/bash
franja:x:1010:1011:El bueno:/home/franja:/bin/bash
```

¿Cuál crees que debería ser la salida del programa?

```
LISTA DE USUARIOS CORRECTOS
--------------------------
Total de usuarios correctos: 6
 1: usuario= amalia id_usuario= 1002    id_grupo=1002   directorio_inicio=/home/amalia
 2: usuario= david  id_usuario= 520     id_grupo=520    directorio_inicio=/home/david
 3: usuario= franja id_usuario= 1010    id_grupo=1011   directorio_inicio=/home/franja
 4: usuario= linoadmin  id_usuario= 1000    id_grupo=1000   directorio_inicio=/home/linoadmin
 5: usuario= root   id_usuario= 0       id_grupo=0      directorio_inicio=/root
 6: usuario= usuario1   id_usuario= 500         id_grupo=501    directorio_inicio=/home/usuario1
```

Recomendación:

La manera más sencilla de almacenar las cuentas e indexarlas por el nombre de usuario sería algún tipo de estructura `Map` donde las **claves** fueran los **nombres de los usuarios**. De esa manera no podrás tener más de una entrada por un mismo nombre de usuario. Por otro lado, si eliges una implementación de `Map` apropiada dentro de las disponibles (`HashMap`, `TreeMap`, `LinkedHashap`, etc.) en la que los objetos se vayan ordenando por la clave según se introduzca un nuevo elemento, tendrías resuelta automáticamente la cuestión de la ordenación.

La estructura de datos Map almacena un par de elementos <clave>:<datos>. De esta forma, como almacenamos bastante información adicional (id de usuario, id de grupo, etc.) deberemos crear un objeto Usuario, donde se almacene todos los atributos de cada usuario.

De esta forma, los tipos que almacena el Map serán <String,Usuario>.

2 Archivo log o de registro

Añadir al programa ValidadorUsuarios anterior la **gestión de un archivo de registro** (uso de la clase `Logger`) para llevar un control de los eventos importantes que puedan ir sucediendo durante la ejecución de la aplicación.

Vamos a considerar los siguientes posibles eventos:

- **Aplicación iniciada**, cuando se inicie el programa. Será de tipo "*informativo*" (`Level.INFO`).
- **Aplicación finalizada**, cuando termine el programa. De tipo "*informativo*" (`Level.INFO`).
- **Archivo de cuentas abierto**, cuando se abra correctamente y sin problemas el archivo de cuentas de usuario. De tipo "*informativo*" (`Level.INFO`)
- **Error de E/S al intentar abrir archivo de cuentas** (porque no exista, no se tenga permiso, etc.). De tipo "*grave*" (`Level.SEVERE`).
- **Cuenta de usuario válida,** cuando se procese una línea que represente una cuenta válida. De tipo "*informativo*" (`Level.INFO`).
- **Cuenta de usuario no válida**, cuando se procese una cuenta que represente una cuenta no válida (porque el usuario no sea correcto, porque no se respeten los formatos, etc.). De tipo "*aviso*" (`Level.WARNING`).

Entre cada registro de un evento debe dejarse una línea en blanco para que sea fácilmente distinguible cada uno. Cuando el programa finalice deben dejarse tres o cuatro líneas en blanco adicionales para distinguir con facilidad cuándo comienza el registro de una nueva ejecución del programa.

El archivo log o de registro se deberá llamar `filtrador.log` y se encontrará en la misma ubicación en que esté la propia aplicación (su path será por tanto `./filtrador.log`). El formato del registro será de tipo **archivo de texto simple** (`SimpleFormatter`) y se debe conservar para todas las ejecuciones que se vayan realizando y no reiniciar cada vez que se arranque de nuevo la aplicación. Esa es la razón por la que se pide que se dejen varias líneas en blanco después del evento de finalización de la ejecución.

Ejemplo de ejecución

De esta forma, si el archivo de entrada tiene los datos anteriores. ¿Cuál sería la salida?

```
mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Aplicación iniciada

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Archivo abierto: etc-passwd.txt

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Cuenta de usuario válida. Línea aceptada: root

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
ADVERTENCIA: Cuenta de usuario no válida. Línea rechazada: amalia

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
ADVERTENCIA: Cuenta de usuario no válida. Línea rechazada:

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Cuenta de usuario válida. Línea aceptada: linoadmin

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Cuenta de usuario válida. Línea aceptada: franja

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
ADVERTENCIA: Cuenta de usuario no válida. Línea rechazada: franja

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Cuenta de usuario válida. Línea aceptada: amalia

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Cuenta de usuario válida. Línea aceptada: david

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
ADVERTENCIA: Cuenta de usuario no válida. Línea rechazada:

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
ADVERTENCIA: Cuenta de usuario no válida. Línea rechazada: linoadmin

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
ADVERTENCIA: Cuenta de usuario no válida. Línea rechazada:

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
ADVERTENCIA: Cuenta de usuario no válida. Línea rechazada: tom

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
ADVERTENCIA: Cuenta de usuario no válida. Línea rechazada:

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
ADVERTENCIA: Cuenta de usuario no válida. Línea rechazada:

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
ADVERTENCIA: Cuenta de usuario no válida. Línea rechazada:

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Cuenta de usuario válida. Línea aceptada: usuario1

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Cuenta de usuario válida. Línea aceptada: pepe

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Cuenta de usuario válida. Línea aceptada: franja

mar. 01, 2022 7:42:35 P. M. ejercicio2.ValidadorUsuarios registrarEvento
INFORMACIÓN: Aplicación finalizada
```

Recomendación:

Para facilitar la implementación de esta funcionalidad te recomendamos que escribas un método específico para registrar los eventos en el log y lo invoques cada vez que se produzca algún suceso digno de ser registrado. De esta manera, tan solo irás añadiendo una única línea a tu código principal que desarrollaste en el apartado anterior y el resto de la carga lo tendrás en ese método que podrías llamar, por ejemplo, `registrarEvento`.

También te recomendamos la definición de un `enum` para especificar cada uno de los eventos que quieras registrar:

```
public enum Evento {
         OPEN_FILE_DONE, OPEN_FILE_FAIL, VALID_ACCOUNT, INVALID_ACCOUNT,
         APP_OPEN, APP_QUIT
}
```

De esta manera, cada vez que se produzca un evento. Por ejemplo, si se ha abierto el archivo de cuentas con éxito, podrías hacer algo así como:

```
registrarEvento (Evento.OPEN_FILE_DONE, archivo.getName());
```

Y ya será el método `registrarEvento` quien se encargue de escribir en el archivo log todo lo que haga falta en función del tipo de evento que le estés mandando. Aquí tienes un ejemplo de cómo podrías hacerlo:

```
public static void registrarEvento (Evento evento,  ...
      switch (evento) {
         case OPEN_FILE_DONE:
              // Registrar evento de archivo abierto
              ...
              break;
         case OPEN_FILE_FAIL:
              // Registrar evento de error de E/S al intentar abrir archivo
              ...
              break;
         case VALID_ACCOUNT:
              // Registrar evento de cuenta de usuario válida
              ...
              break;
         case INVALID_ACCOUNT:
              // Registrar evento de cuenta de usuario no válida
              ...
```

```
            break;


        case ...
```

3 Evaluación de la tarea

Cristerios de evaluación implicados

a) Se han identificado y aplicado principios y prácticas de programación segura.

b) Se han analizado las principales técnicas y prácticas criptográficas.

c) Se han definido e implantado políticas de seguridad para limitar y controlar el acceso de los usuarios a las aplicaciones desarrolladas.

d) Se han utilizado esquemas de seguridad basados en roles.

e) Se han empleado algoritmos criptográficos para proteger el acceso a la información almacenada.

f) Se han identificado métodos para asegurar la información transmitida.

g) Se han desarrollado aplicaciones que utilicen sockets seguros para la transmisión de información.

h) Se han depurado y documentado las aplicaciones desarrolladas.

Nuestros programas no pueden abortar porque se produzca algún error inesperado que haga que salte alguna excepción no controlada que no capturemos y finalmente sea capturada por la máquina virtual de Java haciendo que el programa termine de manera abrupta.

¡ESO NO PUEDE SUCEDER!

Debes ser especialmente cuidadoso con:

- El control de posibles **errores con los parámetros y valores de entrada**: no superar los límites del array `args`, controlar si los parámetros cumplen los criterios especificados (por ejemplo si deben ser números: excepciones del tipo `NumberFormatException`, `InputMismatchException`), etc.
- **Errores de E/S** en la gestión de archivos, sockets, flujos, etc.
- En general cualquier error que nosotros intuyamos que pueda producirse durante la ejecución de nuestro programa.

El hecho de que nuestros programas presenten este tipo de fragilidades y se "rompan" con esa facilidad eclipsa el buen hacer que hayamos podido desarrollar durante la implementación de nuestro código. Si durante la presentación de una aplicación, ésta falla nada más empezar vamos a producir una muy mala impresión a nuestros clientes, jefes, alumnos, profesores, asistentes, etc. echando por tierra todo el esfuerzo que hayamos dedicado a nuestro trabajo. Hay que cuidar este tipo de detalles en el acabado de nuestro producto.

Este tipo de fallos será severamente penalizado.

En la siguiente tabla se muestran los criterios de corrección aplicables a cada ejercicio, y la puntuación máxima asignada al mismo.

Criterios de corrección	*Puntuación*
• **Se recibe como parámetro desde la línea de órdenes (*command line*) en consola el archivo de cuentas que va a ser filtrado.** • Se llevan a cabo las **comprobaciones apropiadas con el parámetro recibido**. • Se **gestionan apropiadamente los posibles errores** en el tratamiento del archivo de entrada. • Se **recorre y analiza apropiadamente cada línea** del archivo de entrada. • Se **admiten las cuentas válidas** y se **descartan las inválidas**. • **Si un usuario se repite**, siempre nos quedaremos con la última ocurrencia válida. • Se muestra por pantalla la **lista de cuentas válidas ordenada alfabéticamente por nombre de usuario** y con el **formato indicado** en el enunciado. • La **ordenación de las cuentas** se realiza de una manera sencilla y sin complicaciones innecesarias. • No contiene **elementos extraños**, **redundantes** o **sin sentido**. • No se "rompe" con facilidad.	Hasta 5 puntos
• Se genera y configura un **archivo log de registro de eventos** tal y como se indica en el enunciado. • Se registra apropiadamente cada uno de los eventos indicados en el enunciado: ◦ **Aplicación iniciada**, ◦ **Aplicación finalizada**. ◦ **Archivo de cuentas abierto**.	Hasta 5 puntos

◦ **Error de E/S al intentar abrir archivo de cuentas** (porque no exista, no se tenga permiso, etc.). ◦ **Cuenta de usuario válida**. ◦ **Cuenta de usuario no válida** • No contiene **elementos extraños**, **redundantes** o **sin sentido**. • No se "rompe" con facilidad.	