PRG (E.T.S. de Ingeniería Informática) - Curso 2018-2019

Práctica 4. Tratamiento de excepciones y ficheros en un registro ordenado de accidentes. Segunda parte

(1 sesión)

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Índice

1. Contexto y trabajo previo

En el marco académico, esta práctica corresponde al "*Tema 3. Elementos de la POO: herencia y tratamiento de excepciones*" y al "*Tema 4. E/S: ficheros y flujos*". El objetivo principal que se pretende alcanzar con ella es que refuerces y pongas en práctica los conceptos introducidos en las clases de teoría sobre el tratamiento de excepciones y la gestión de la E/S mediante ficheros y flujos. En particular, al finalizar esta práctica debes ser capaz de:

- Lanzar, propagar y capturar excepciones local y remotamente.
- Leer/escribir desde/en un fichero de texto.
- Tratar las excepciones relacionadas con la E/S.

Para ello, durante las tres sesiones de prácticas, se va a desarrollar una pequeña aplicación en la que se procesarán los datos que se lean de ficheros de texto, guardando el resultado en otro fichero.

2. Planteamiento del problema

En esta segunda parte de la práctica, utilizando las clases `CorrectReading` y `SortedRegister` de la primera parte, se va a desarrollar una pequeña aplicación, `SortFiles`, que abrirá todos los ficheros de datos que se encuentren en un directorio del sistema, y los agregará ordenándolos cronológicamente en un fichero de resultados.

Por ejemplo, en la figura 1 se muestra la carpeta que se ha creado dentro del proyecto `prg` de prácticas, en el paquete `pract4` y dentro de la carpeta `data` en la que se han copiado unos ficheros de datos.

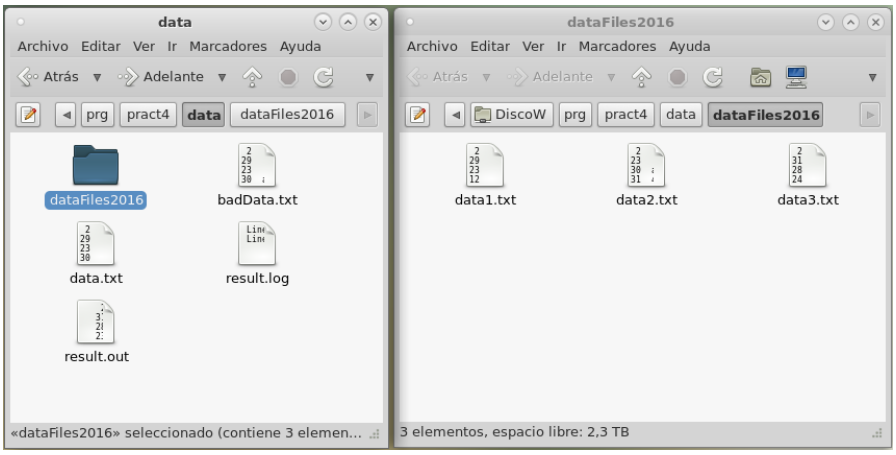

Figura 1: Directorio `dataFiles2016` con los ficheros `data1.txt`, `data2.txt` y `data3.txt`.

La aplicación crea una carpeta en el mismo lugar a propósito para guardar los ficheros que resultaran de procesar los datos: el fichero de resultados `result.out` y el fichero de informe de errores `result.log`, como se ve en el ejemplo de la figura 2.

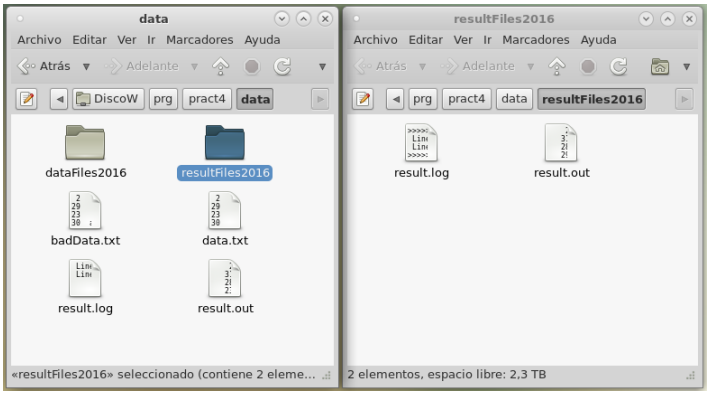

Figura 2: Directorio `resultFiles2016` con los ficheros de resultados `result.out` y de errores `result.log`.

El procesamiento realizado por `SortFiles` sólo se completará si es posible acceder correctamente a todos los ficheros implicados.

3. Un ejemplo de uso de la clase `File`

3.1. Actividad 1: descarga de ficheros en `pract4`

- Descargar, desde `Recursos/Laboratorio/Práctica 4` de *PoliformaT* de `PRG`, el fichero `SortFiles.java`, y agregarlo al paquete `pract4`.
- Crear dentro del directorio `prg/pract4/data` una nueva carpeta `dataFiles2016`. Descargar, desde el mismo sitio, los ficheros `data1.txt`, `data2.txt` y `data3.txt`, y copiarlos en `dataFiles2016`.

3.2. Actividad 2: prueba de la clase `SortFiles`

La clase `SortFiles` supone que, dado un año `AAAA` para el que se dispone de datos, se ha creado en el sistema, previamente a su ejecución, una carpeta de nombre `dataFilesAAAA` que contiene todos los ficheros con los datos correspondientes.

Para procesar tales datos, la clase contiene dos métodos:

- Método `main`, que se encarga de reconocer el directorio del sistema que contiene los ficheros de datos y de crear el directorio en donde se dejarán los resultados. Para ello, le pide al usuario que introduzca un año `AAAA` y la ubicación del directorio con los ficheros a ordenar (se guarda en la variable `dirParent`). Entonces, crea un descriptor de fichero para `dirParent/dataFilesAAAA`, con lo que se puede comprobar si existe tal directorio en el sistema.

  En tal caso, se obtiene el array con los descriptores de todos los ficheros que contuviese, para pasar a procesarlos y guardar los resultados en la carpeta `dirParent/resultFilesAAAA` (invocando al método `reportedSortFiles(File[], int, File)` siguiente). En caso contrario, se avisa de que dicho directorio no existe.
- Método `reportedSortFiles(File[], int, File)`, que se encargará de crear el `SortedRegister` en el que acumular los datos que se lean de todos los ficheros y con el que escribir los datos en los ficheros de resultados y errores. Recibe como parámetros la lista de descriptores de los ficheros de datos, el año al que corresponden dichos datos y el descriptor de la carpeta en la que se guardarán los resultados.

  El cuerpo del método se ha dejado vacío y se deberá completar en las actividades del siguiente apartado. Por lo que, a falta de completar su código, la llamada realizada por `main` a dicho método no produce ningún resultado, y la carpeta creada para los resultados queda vacía.

El método `main` de la clase es un ejemplo de cómo usar la clase `File` para describir rutas en el sistema de archivos, e incluso modificarlo. En esta actividad, se probará a ejecutar la aplicación usando la carpeta `dataFiles2016` creada en la actividad anterior. Se introducirá por teclado:

- El año 2016.
- La ubicación de la carpeta `dataFiles2016`. Esta se encuentra dentro del directorio `data` que, a su vez, está dentro del paquete `pract4` en el que se está ejecutando la clase. Por tanto, se debe introducir como ruta relativa el propio nombre del directorio: `data`, tal como se muestra en la figura 3.

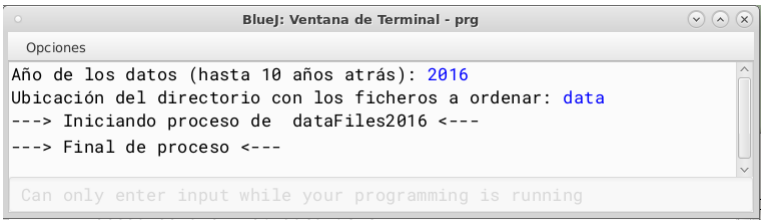

Figura 3: Ejemplo de ejecución de `SortFiles` con `reportedSortFiles` incompleto.

Como resultado de esta ejecución de prueba preliminar, se debe de haber generado en `prg` la carpeta `resultFiles2016` (vacía) en caso de que no existiese previamente.

4. Excepciones de acceso a fichero

Como se ha descrito en el apartado anterior, la clase `SortFiles` contiene un método que va a ser el encargado de leer los datos del problema y generar los resultados, y cuyo perfil es:

```
public static void reportedSortFiles(File[] listF, int year, File place)
```

Cabe notar que el método recibe com primer parámetro únicamente los descriptores de los ficheros, y será el encargado de abrir unos para su lectura, y crear otros en la ubicación indicada por `place` para escribir en ellos, o sobrescribirlos si ya existiesen. Será igualmente el encargado de resolver localmente las excepciones comprobadas que estas operaciones comportan.

En la llamada desde el `main` a este método, el primer argumento es `f.listFiles()`. Cuando se invoca al método `listFiles()` sobre un objeto de tipo `File` y dicho objeto representa a un directorio, el resultado es un array de `File` pero, según la documentación de la clase `File`, dicho array NO está ordenado, en concreto, la documentación de *Java* dice: "*There is no guarantee that the name strings in the resulting array will appear in any specific order; they are not, in particular, guaranteed to appear in alphabetical order.*"

Por ejemplo, en las figuras 4 y 5 se muestran los resultados de ejecutar la aplicación para el año 2016 con los ficheros `data1.txt`, `data2.txt` y `data3.txt` de `dataFiles2016`.

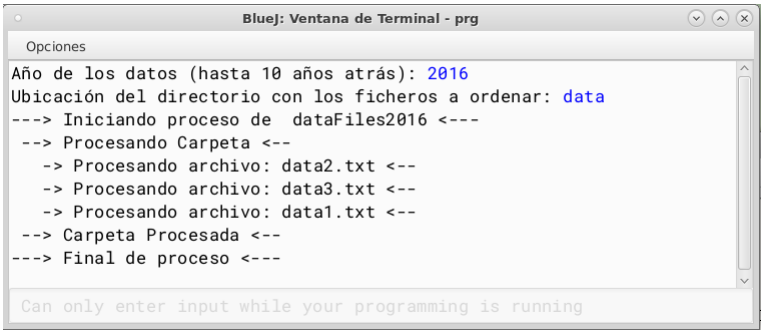

Figura 4: Ejemplo de ejecución de `SortFiles` con `reportedSortFiles` completo.

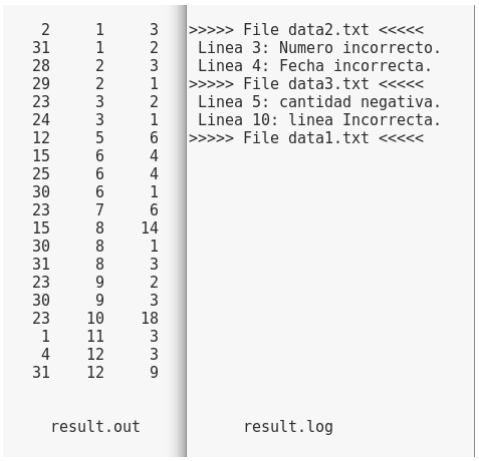

Figura 5: Resultados de `SortFiles` con los ficheros de `dataFiles2016`.

4.1. Actividad 3: desarrollo del método `reportedSortFiles`

La excepción `FileNotFoundException` de apertura de los ficheros es comprobada, por lo que es obligatorio propagarla explícitamente en la cabecera del método o tratarla. Este método no la propagará; el tratamiento consistirá en informar al usuario de qué fichero ha dado problemas de acceso y ha impedido completar el proceso.

Para ello, en el cuerpo del método, se harán las siguientes declaraciones iniciales:

```
Scanner in = null;
PrintWriter out = null, error = null;
File f = null;
String fName = "";
```

y se deberá intentar en un bloque `try` la ejecución del código que implemente lo siguiente:

1. Crear el `SortedRegisted c` para el año `year`, que se usará para clasificar ordenadamente todos los datos.
2. A partir del descriptor `f = new File(place + "/result.log")`, abrir para escritura el fichero `error`.
3. Entonces, se deberán abrir para lectura todos los ficheros de `listF`, acumulando los datos en `c`, y registrando en `error` los eventuales errores:

```
for (int i = 0; i < listF.length; i++) {
    f = listF[i];
    fName = f.getName();
    in = new Scanner(f);
    System.out.println("  --> Procesando archivo: " + fName +  " <--");
    error.println(">>>>> File " + fName + " <<<<<");
    c.add(in, error);
    in.close();
}
```

   Notar que todo fichero de la lista se va cerrando a medida que se completa su procesamiento.
4. A partir del descriptor `f = new File(place + "/result.out")`, abrir para escritura el fichero `out`, salvar en él la información clasificada en `c`, y dejarlo cerrado.

El bloque `catch` debe capturar la excepción `FileNotFoundException` que puede producirse al abrir cualquiera de los ficheros anteriores. Dado que, en ese caso, la variable `f` conserva el descriptor del fichero que ha producido la excepción, se informará al usuario escribiendo en la salida el mensaje `"Proceso incompleto: Error al abrir " + f`, y con ello, se dará por finalizado el proceso.

Hay que remarcar que, de producirse la excepción, queda garantizado que todos los ficheros que se hubieran abierto anteriormente se habrían terminado por cerrar, con la única excepción de `error`, por lo que en un bloque `finally` se comprobará si `error` se ha podido abrir sin producir niguna excepción (es diferente de `null`), y en tal caso se cerrará.

4.2. Actividad 4: prueba del método `reportedSortFiles`

Una vez completado el método, se harán las siguientes pruebas:

- Ejecutar `main`, introduciendo por teclado los datos de la figura 4, y comprobar que los ficheros resultantes coinciden con los de la figura 5.
- Para probar una ejecución en la que se dé la excepción `FileNotFoundException`, cambiar, por ejemplo, los permisos de acceso del fichero `result.out` resultante de la ejecución anterior. Para esto, pinchar el icono del fichero con el botón derecho del ratón, seleccionar las propiedades, y dejarlo únicamente con permiso de lectura, como se ve en la figura 6.

  Repetir la ejecución anterior y comprobar que se escribe en la salida el mensaje de error:

  ```
  Proceso incompleto: Error al abrir pract4/data/resultFiles2016/result.out
  ```

- Repetir la ejecución habiendo quitando el permiso de escritura a `result.log`, para comprobar que cuando falla la apertura de `error`, se tiene en cuenta correctamente en el bloque `finally` que `error` no se debe cerrar.

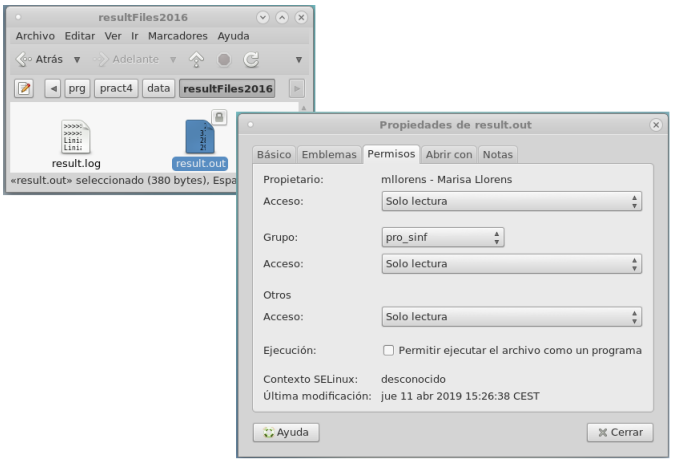

Figura 6: Cambio de permisos de acceso de `result.out`.

5. Evaluación

Esta práctica forma parte del segundo bloque de prácticas de la asignatura que será evaluado en el segundo parcial de la misma. El valor de dicho bloque es de un 60 % con respecto al total de las prácticas. El valor porcentual de las prácticas en la asignatura es de un 20 % de su nota final.