

PRG (E.T.S. d'Enginyeria Informàtica) - Curs 2019-2020
Pràctica 4. Tractament d'excepcions i fitxers
Segona part: Obtenció d'un registre ordenat d'accidents a partir
d'un fitxer de dades.
(Dues sessions)

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València



Índex

1	Plantejament del problema	1
2	Les classes de l'aplicació	2
2.1	Activitat 1: instal·lació de les classes i fitxers de prova	3
2.2	Activitat 2: examen i prova de la classe <code>SortedRegister</code>	3
3	Gestió d'excepcions <code>RuntimeException</code>	5
3.1	Activitat 3: examen i prova del mètode <code>handleLine(String)</code>	5
3.2	Activitat 4: finalització del mètode <code>handleLine(String)</code>	5
3.3	Activitat 5: captura d'excepcions en el mètode <code>add(Scanner)</code>	6
3.4	Activitat 6: desenvolupament del mètode <code>add(Scanner, PrintWriter)</code>	8
4	Gestió d'excepcions <code>IOException</code>	9
4.1	Activitat 7: captura d'excepcions de classe <code>FileNotFoundException</code> en el programa <code>TestSortedRegister</code>	10

1 Plantejament del problema

Es disposa d'un registre del nombre d'accidents esdevinguts al llarg d'un any determinat. El registre pot provenir d'una o més àrees (ciutats, províncies, ...), i les dades poden trobar-se distribuïdes en un o més fitxers de text, on cada línia té el format següent:

`dia mes quantitat`

sent `dia` i `mes` uns enters majors que 0 corresponents a una data de l'any, i `quantitat` és un enter no negatiu corresponent a una dada d'accidents registrats en aqueixa data.

En un mateix fitxer de dades poden donar-se dates repetides, i les línies no tenen per què estar en ordre cronològic, com podria donar-se si en un mateix fitxer s'hagueren concatenat les dades procedents de diverses àrees.

Es desitja una aplicació que extraga les dades d'un any a partir d'un o més fitxers de text, i genere un fitxer de resultats en el qual apareguen registrades, i per ordre cronològic, les dades acumulades de cada data per a la qual consten registres, a la manera que es mostra en la figura 1. Ha de contemplar-se també, la possibilitat que els fitxers continguin anotacions errònies, bé perquè una línia conté més o menys de tres valors o aquests no siguin enters, perquè **dia** i **mes** no siguin una data correcta, o perquè **quantitat** siga negativa.

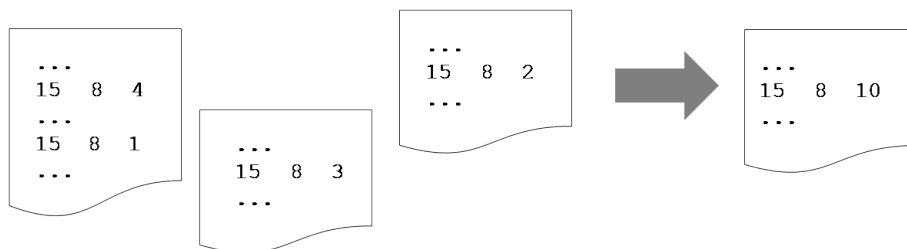


Figura 1: Agregació de dades procedents d'un o diversos fitxers.

Per a resoldre el problema s'usarà una matriu les files de la qual vinguen indexades per mesos, i les columnes vinguen indexades pels dies de cada mes, de manera que la quantitat que aparega en cada línia de dades que es processe, s'acumuli directament en la component indexada pel mes i el dia de la data. Una vegada bolcades les dades en la matriu, un recorregut per files i columnes permet obtenir un llistat, ordenat per dates, de les dades que s'hagueren acumulat.

2 Les classes de l'aplicació

En el material de la pràctica es proporciona la classe tipus de dades **SortedRegister**, la funcionalitat del qual permet el processament de dades com les anteriors.

Els objectes de la classe **SortedRegister** (veure figura 2) contenen un array bidimensional **m** en el qual les files són mesos i les columnes els dies de cada mes, de manera que **m[f][c]** s'usarà per a emmagatzemar els accidents acumulats el dia **c** del mes **f**. Les files 1 a 12 es fan correspondre als mesos de l'any (la fila 0 no s'utilitzarà). Per a cada fila, les columnes de la 1 endavant correspondran als dies del mes (la columna 0 no s'utilitzarà). Notar que l'última columna del mes de febrer haurà de ser la 28 o la 29, depenent que l'any siga o no bixest. Els principals mètodes que vénen implementats en la classe són:

- El constructor

```
public SortedRegister(int year)
```

que crea la matriu **this.m** d'acord amb l'any donat per **year**.

- Mètode de perfil

```
public int add(Scanner sc)
```

que, sent **sc** un **Scanner** obert a partir de la font de text de les dades, processa les línies de **sc** per a bolcar les dades en **this.m**. Si el procés acaba normalment (sense produir-se excepcions per format incorrecte en les dades), retorna el nombre de línies processades.

- Mètode de perfil

```
public void save(PrintWriter pw)
```

que escriu en `pw` les dades acumulades en `this.m`, per ordre cronològic.

Per a fer proves del comportament de `SortedRegister`, llegint les dades d'uns fitxers concrets, es disposa de la classe programa `TestSortedRegister`. En aquesta classe s'usa també la classe d'utilitats `CorrectReading` desenvolupada en la primera part de la pràctica.

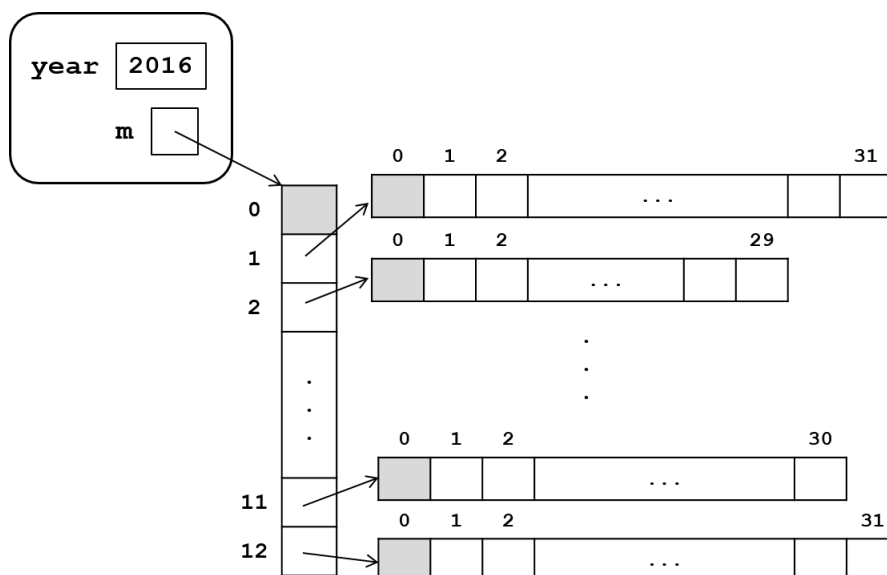


Figura 2: Estructura d'un `SortedRegister`.

2.1 Activitat 1: instal·lació de les classes i fitxers de prova

- Descarregar de la carpeta de material de la pràctica 4, els fitxers `SortedRegister.java` i `TestSortedRegister.java` i agregar-los al paquet `pract4`.
- Des de l'explorador d'arxius i dins de la carpeta `prg/pract4` crear una nova carpeta amb el nom `data` on es deixaran i crearan els fitxers de dades i resultats.
- Descarregar del mateix lloc els fitxers de text que s'usaran per a fer proves: `data.txt`, `badData1.txt`, `badData2.txt`, `badData3.txt`, `badData4.txt`. Copiar-los en la carpeta `prg/pract4/data`.

És important que els fitxers de dades se situen en la ubicació correcta: com es podrà comprovar en la següent activitat, el codi del programa `TestSortedRegister` cerca els fitxers amb els quals fer les proves en la subcarpeta `pract4/data` del projecte `prg`. Els fitxers de resultats els deixa en la mateixa carpeta `data`.

2.2 Activitat 2: examen i prova de la classe `SortedRegister`

Examinar el codi de la classe `SortedRegister` que s'ha descarregat de *PoliformaT*: estructura de la matriu de dades, mètode `add` (que usa un mètode auxiliar `handleLine` per a obtenir les dades de cada línia i actualitzar amb ells la matriu `this.m`), i mètode `save`.

Examinar també la classe `TestSortedRegister`, que conté un mètode `main` i un mètode `test1` que serveix per a provar els mètodes de `SortedRegister`. El mètode `main` s'encarrega

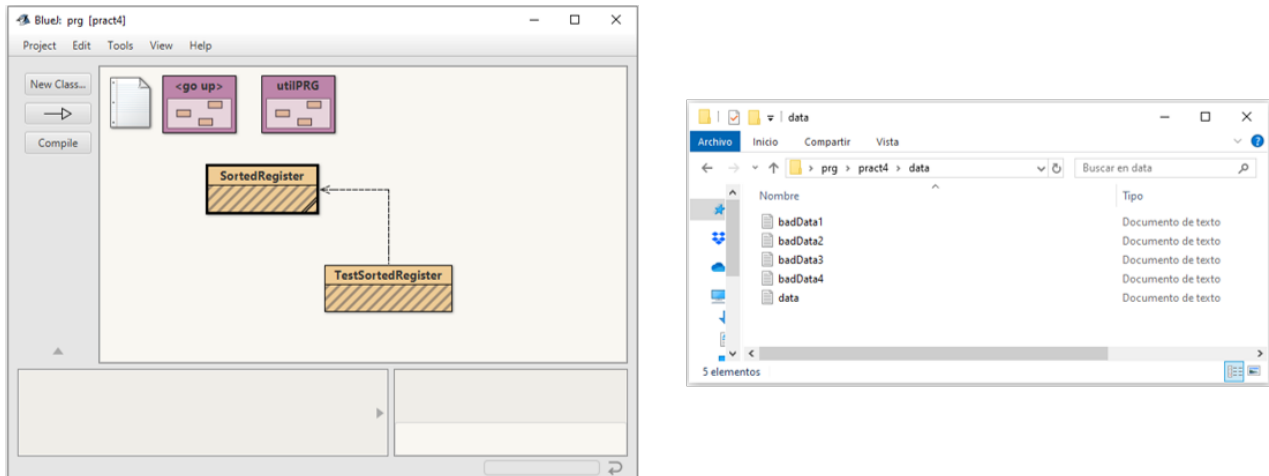


Figura 3: Paquet `pract4` amb les classes de l'aplicació i carpeta `data` amb els fitxers de dades.

de llegir un any correcte dins d'un interval i el nom d'un fitxer de dades, obrir un `Scanner` i un `PrintWriter` a partir dels fitxers de dades i de resultats (`result.out`), respectivament, i usar el mètode `test1` per a realitzar el processament de les dades.

Provar d'executar la classe introduint com a dades l'any 2016 (bixest), el fitxer `data.txt`, i seleccionant l'opció 1 del menú (`test1`). Comparar el contingut del fitxer introduït amb el creat com a resultat de l'execució. Com es mostra en la figura 4, es pot observar que en `result.out` apareixen les dades de `data.txt`, convenientment acumulades, i en ordre cronològic.

<code>data.txt</code>			<code>result.out</code>		
2	1	1	2	1	2
29	2	1	31	1	2
23	3	1	28	2	3
30	4	2	29	2	1
12	5	2	23	3	1
25	6	2	24	3	1
15	8	3	30	4	1
30	8	1	12	5	4
23	9	2	15	6	4
30	9	1	25	6	2
23	10	6	30	6	1
1	11	3	23	7	6
4	12	1	31	7	1
31	12	3	15	8	10
2	1	1	30	8	1
31	1	2	31	8	3
28	2	3	23	9	2
24	3	1	30	9	2
30	4	-1	23	10	12
12	5	2	1	11	3
15	6	4	4	12	2
30	6	1	31	12	6
23	7	6			
31	7	1			
15	8	7			
31	8	3			
30	9	1			
23	10	6			
4	12	1			
31	12	3			

Figura 4: Contingut dels fitxers de text `data.txt` i `result.out`.

Atenció: en executar el mètode `main` de `TestSortedRegister` es poden produir excepcions d'entrada/eixida si s'escriu malament el nom del fitxer, o no es troba en la ubicació esperada; aquestes excepcions es tractaran més endavant, en la secció 4.

3 Gestió d'excepcions `RuntimeException`

3.1 Activitat 3: examen i prova del mètode `handleLine(String)`

Per a processar cadascuna de les línies que el mètode `add` llig amb un `nextLine()`, invoca al mètode privat auxiliar `handleLine(String)` passant-li la línia com a paràmetre.

Si examinem el codi d'aquest mètode auxiliar, veiem que el primer que fa és aplicar a `line` el mètode `split` de `String`; això es fa per a “dividir” la línia en els substring que apareguen separats entre si per blancs: `split` retorna un array les components del qual són els successius substrings en els quals es divideix la línia, de manera que si `line` s'ajusta al format l'array resultant tindrà exactament tres components.

Si s'han pogut extraure correctament els enters `day`, `month`, `amount`, la quantitat llegida s'emmagatzema en la component `this.m[month][day]` de la matriu.

Notar que el mètode propaga les excepcions corresponents a les següents situacions d'error:

1. Si la línia dividida mitjançant el mètode `split` no té exactament tres dades, es crea i es llança una excepció de classe `IllegalArgumentException` amb el missatge "La linia no conte tres dades."
2. Si alguna de les tres dades de la línia no es pot transformar a `int`, es propaga l'excepció `NumberFormatException` deguda al mètode `Integer.parseInt`.
3. Si `day` i `month` no són índexs ≥ 1 corresponents a una component de la matriu `this.m`, es crea i es llança una excepció `IllegalArgumentException` amb el missatge "Data incorrecta.". Cal recordar que el constructor de `SortedRegister` crea la matriu `this.m` amb les files 1 a 12 corresponents als mesos de l'any, i per a cadascuna d'aquestes files, les seues columnes des de la 1 endavant, corresponents als dies del mes.

Per a les situacions 1 i 3 anteriors, s'ha triat crear una excepció no comprovada de classe `IllegalArgumentException` l'ús de la qual, com es veu en la documentació de la figura 5, està indicat quan a un mètode se li passa un argument inapropiat. El missatge amb què es crea permet distingir entre aquestes dues situacions.

Per a comprovar la propagació de les excepcions a `add`, i d'ací al programa, es farà la següent prova:

Tornar a executar el `main` de la classe `TestSortedRegister`, opció 1, però passant com a dades l'any 2016 i el fitxer `badData1.txt`. Observar quina és l'excepció que es produeix (propagada per `handleLine`).

A continuació, examinar el contingut de `badData1.txt`, i comprovar que la primera línia errònia que conté el fitxer es correspon amb aquesta excepció.

Comprovar a més que `result.out` ha quedat buit, atès que `test1` s'interromp sense aconseguir la instrucció d'escriptura en el fitxer de resultats.

3.2 Activitat 4: finalització del mètode `handleLine(String)`

El mètode `handleLine` que s'ha examinat i provat en l'activitat anterior està incomplet. En efecte, reconsiderar l'execució de prova de l'activitat 3 anterior, i el resultat de la qual es mostrava en la figura 4. S'aprecia que en `data.txt` apareixen les següents dues línies per al 30 d'abril:



Figura 5: Fragment de la documentació d'`IllegalArgumentException`.

```

....
30  4  2
....
30  4  -1
....

```

El processament d'ambdues ha produït en `result.out` la línia resultant de sumar les quantitats 2 i -1:

```

30  4  1

```

És a dir, el mètode `handleLine` ha oblidat la detecció d'una quantitat errònia (el nombre d'accidents esdevinguts en una data no pot ser menor que 0).

Així doncs, perquè `handleLine` contemple tots els errors possibles, afegir en el mètode una instrucció que, abans de passar a emmagatzemar en la matriu la quantitat llegida, comprovi si és negativa, i en aqueix cas cree i llance una excepció `IllegalArgumentException` amb el missatge "Quantitat negativa."

Tornar a executar el `main` de la classe `TestSortedRegister`, opció 1, passant de nou com a dades l'any 2016 i el fitxer `data.txt`. Comprovar que, quan es processe la línia amb la quantitat negativa, es produïska l'excepció corresponent (figura 6).

3.3 Activitat 5: captura d'excepcions en el mètode `add(Scanner)`

El mètode `add` implementat en la classe té com a precondition que les línies que es lligen del `Scanner` s'ajusten al format establert. Com s'ha vist en les activitats anteriors, si es llig una línia incorrecta, el mètode es limita a propagar l'excepció que li arriba de `handleLine`; en el programa de prova `TestSortedRegister`, això provoca que els mètodes `test1` i `main` les propaguen al seu torn, amb el que s'acaba abruptament l'execució.

Desitgem refinar el comportament del mètode perquè acabe normalment (sense propagar cap excepció) en qualsevol cas, de manera que:

- Si totes les línies són correctes, el mètode retornarà el nombre de línies processades.

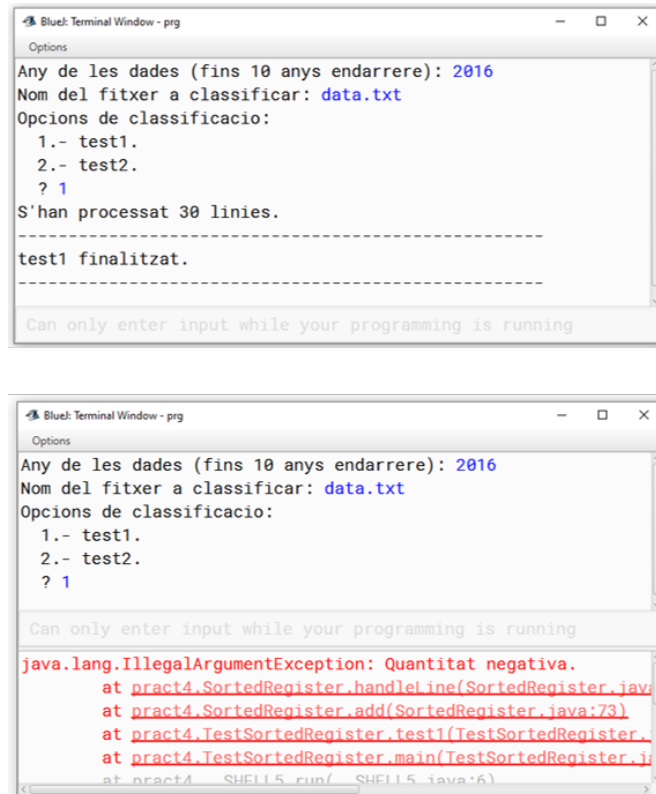


Figura 6: Creació i propagació de l'excepció deguda a una quantitat < 0 : l'execució de dalt és prèvia a completar `handleLine` segons l'activitat 4, i la de baix és posterior.

- En cas contrari, quan detecte que la font del **Scanner** té alguna línia amb defecte de format, interromprà el processament de les dades i acabarà retornant -1 . Abans d'acabar, haurà d'escriure en l'eixida estàndard, un dels següents missatges, segons l'error detectat:

```
ERROR. Línia n: La línia no conte tres dades.
ERROR. Línia n: Dada no entera.
ERROR. Línia n: Data incorrecta.
ERROR. Línia n: Quantitat negativa.
```

sent n el número de línia en el qual s'ha detectat l'error.

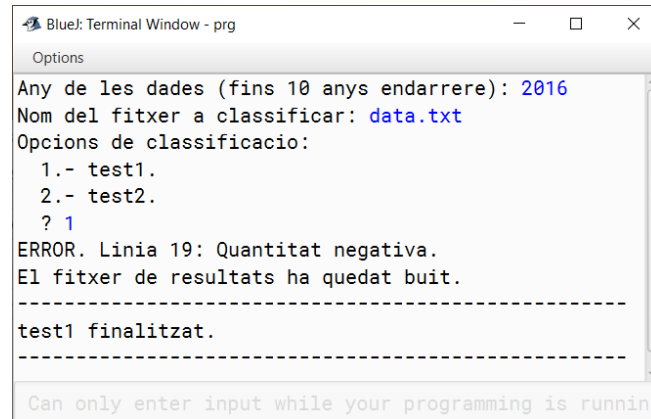
Per a això, el cos del mètode haurà de seguir una estructura com la que es mostra a continuació.

```
int count = 0;
try {
    ...
} catch (NumberFormatException e) {
    System.out.println("ERROR. Línia " + count + ": Dada no entera.");
    count = -1;
} catch (IllegalArgumentException e) {
    System.out.println("ERROR. Línia " + count + ": " + e.getMessage());
    count = -1;
}
return count;
```

És important tenir en compte que l'excepció `NumberFormatException` és una derivada de `IllegalArgumentException` (veure figura 5) pel que el compilador exigeix que el `catch` de l'excepció derivada estiga situat **abans** que el de la seua classe base.

Els comentaris de documentació del mètode hauran de reflectir aquests canvis (indicar que el mètode interromp la lectura de dades si detecta una línia errònia i acaba retornant `-1`, i eliminar les frases etiquetades amb `@throws`).

Amb aquesta modificació, la repetició de l'execució del `test1` amb el fitxer `data.txt` acaba normalment, com succeeix en l'exemple de la figura 7. Observar que en el codi de `test1`, si `c.add(sc)` retorna un valor negatiu, llavors no es bolquen les dades de `c` en el `PrintWriter` out (queda buit).



```
BlueJ: Terminal Window - prg
Options
Any de les dades (fins 10 anys endarrere): 2016
Nom del fitxer a classificar: data.txt
Opcions de classificacio:
  1.- test1.
  2.- test2.
  ? 1
ERROR. Línia 19: Quantitat negativa.
El fitxer de resultats ha quedat buit.
-----
test1 finalitzat.
-----
Can only enter input while your programming is running
```

Figura 7: Captura d'excepcions en `add(Scanner)`: si apareix una línia errònia, `add` no propaga l'excepció corresponent.

Per a comprovar el correcte funcionament del mètode s'han deixat els fitxers `badData1.txt`, `badData2.txt`, `badData3.txt` i `badData4.txt`. Els quatre contenen la mateixa informació, amb les mateixes quatre línies defectuoses (línies número 4, 5, 6 i 7), encara que en diferent ordre. D'aquesta manera, la primera línia errònia de cadascun s'ha fet correspondre amb un dels quatre casos d'error. S'haurà de provar l'opció 1 del `main` de `TestSorteRegister`, i observar el missatge d'error que s'escriu:

- Introduint l'any 2016 i el fitxer `badData1.txt`.
- Introduint l'any 2016 i el fitxer `badData2.txt`.
- Introduint l'any 2016 i el fitxer `badData3.txt`.
- Introduint l'any 2016 i el fitxer `badData4.txt`.

Es pot editar `data.txt` per a eliminar la línia incorrecta amb una quantitat negativa i obtenir un fitxer sense errors. Si es repeteix la prova, el mètode ha de processar correctament el fitxer amb les línies restants.

3.4 Activitat 6: desenvolupament del mètode `add(Scanner, PrintWriter)`

El mètode `add` de `SortedRegister` està sobrecarregat pel mètode de perfil

```
public int add(Scanner sc, PrintWriter err)
```

mancant que es complete el seu codi. Com abans, `sc` és la font de les dades, però ara es processaran totes les seues línies, filtrant les errònies, de les quals es deixarà registre en `err`.

En concret, aquest mètode haurà de ser una modificació de l'anterior, de manera que, per a totes i cadascuna de les línies de `sc`:

- Intente obtenir les dades de la línia i acumular la quantitat llegida en la matriu, valent-se del mètode `handleLine`.
- Capture les excepcions produïdes per `handleLine`, escrivint en `err` una de les següents frases segons el cas:

```
ERROR. Linia n: La linia no conte tres dades.
ERROR. Linia n: Dada no entera.
ERROR. Linia n: Data incorrecta.
ERROR. Linia n: Quantitat negativa.
```

sent `n` el número de línia en la qual es produeix l'excepció.

D'aquesta forma, en `this.m` hauran quedat les dades de les línies que s'ajusten al format, i en `err` s'hauran escrit els missatges d'error. El mètode acabarà retornant el nombre total de línies processades, correctes i incorrectes.

Per a provar-ho, completar el mètode `test2` de la classe `TestSortedRegistered`. El mètode haurà de ser anàleg a `test1`, excepte que haurà d'usar el mètode de perfil `add(Scanner, PrintWriter)` completat en aquesta activitat, i l'escriptura en `out` de les dades ja classificades s'haurà d'executar sempre. Una vegada completat `test2`, es podran fer proves com la de la figura 8.

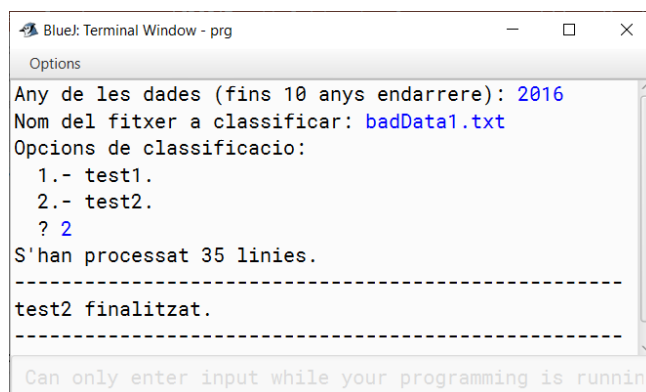


Figura 8: Prova de l'execució d'`add(Scanner, PrintWriter)`.

Per a aquesta execució hauria d'obtenir-se uns fitxers `result.out` i `result.log` com els de la figura 9.

4 Gestió d'excepcions `IOException`

En Java es distingeix entre excepcions *checked* o comprovades, que són de tractament obligat (mitjançant la seua captura o propagació), i excepcions *unchecked* o no comprovades (`RuntimeException` i les seues derivades). Les excepcions comprovades sorgeixen en situacions en les quals normalment no és possible preveure i eludir la fallada, com succeeix típicament en problemes d'accés a fitxers.

En el mètode `main` del programa `TestSortedRegister`, pot succeir l'excepció comprovada `FileNotFoundException` si es dona algun problema en obrir el `Scanner` i els `PrintWriter` que usaran els tests (no es té permís per a fer-ho, no s'introdueix bé el nom del fitxer de dades o no està en la ubicació correcta, no queda suficient memòria per a gravar-los, etc.).

En aquesta secció es desitja capturar aquestes excepcions per a evitar una finalització abrupta del programa.

badData1.txt	result.out	result.log
<pre> 2 1 1 1 6 1 23 3 1 30 1 4 5 30 abril 2 31 9 2 30 4 -1 12 5 2 25 6 2 15 8 3 30 8 1 23 9 2 30 9 1 23 10 6 1 11 3 4 12 1 31 12 3 2 1 1 31 1 2 28 2 3 24 3 1 29 2 1 12 5 2 15 6 4 30 6 1 23 7 6 31 7 1 15 8 7 31 8 3 30 9 1 23 10 6 4 12 1 31 12 3 16 5 1 2 1 2 </pre>	<pre> 2 1 4 31 1 2 28 2 3 29 2 1 23 3 1 24 3 1 12 5 4 16 5 1 1 6 1 15 6 4 25 6 2 30 6 1 23 7 6 31 7 1 15 8 10 30 8 1 31 8 3 23 9 2 30 9 2 23 10 12 1 11 3 4 12 2 31 12 6 </pre>	<pre> ERROR. Línea 4: La línea no conte tres dades. ERROR. Línea 5: Dada no entera. ERROR. Línea 6: Data incorrecta. ERROR. Línea 7: Quantitat negativa. </pre>

Figura 9: Resultat de la prova de la figura 8.

4.1 Activitat 7: captura d'excepcions de classe FileNotFoundException en el programa TestSortedRegister

Examinar el codi del mètode main de TestSortedRegister. Es pot comprovar que en aquest mètode no es realitza cap captura de les possibles excepcions FileNotFoundException, per la qual cosa el compilador obliga al fet que en el perfil del main aparega la clàusula throws FileNotFoundException.

En aquesta activitat s'ha de canviar el codi de main per a no propagar l'excepció que es puga produir. Per a això, una vegada llegits de teclat l'any i el nom del fitxer de les dades, s'haurà d'incorporar la gestió d'excepcions que ve a continuació.

```

...
Scanner in = null; PrintWriter out = null, err = null;
File f = new File("pract4/data/" + nameIn);
try {
    in = new Scanner(f);
    f = new File("pract4/data/" + "result.out");
    out = new PrintWriter(f);
    f = new File("pract4/data/" + "result.log");
    err = new PrintWriter(f);
    ...
    // Selecció i execució del test de prova
    ...
} catch (FileNotFoundException e) {
    System.out.println("Error en obrir el fitxer: " + f);

```

```

} finally {
    if (in != null) { in.close(); }
    if (out != null) { out.close(); }
    if (err != null) { err.close(); }
}

```

És important remarcar que la màquina virtual executa sempre el codi del bloc **finally**. Així assegurem que, en qualsevol circumstància, es tanquen tots els fitxers que hagueren sigut oberts en el bloc **try**.

Una vegada incorporada en el mètode aquesta captura d'excepcions, es pot eliminar la clàusula **throws** de la capçalera de **main**.

En la figura 10 es mostra quin és el comportament de **TestSortedRegister** sense gestió de l'excepció **FileNotFoundException**, i quin ha de ser després d'introduir aquesta gestió.

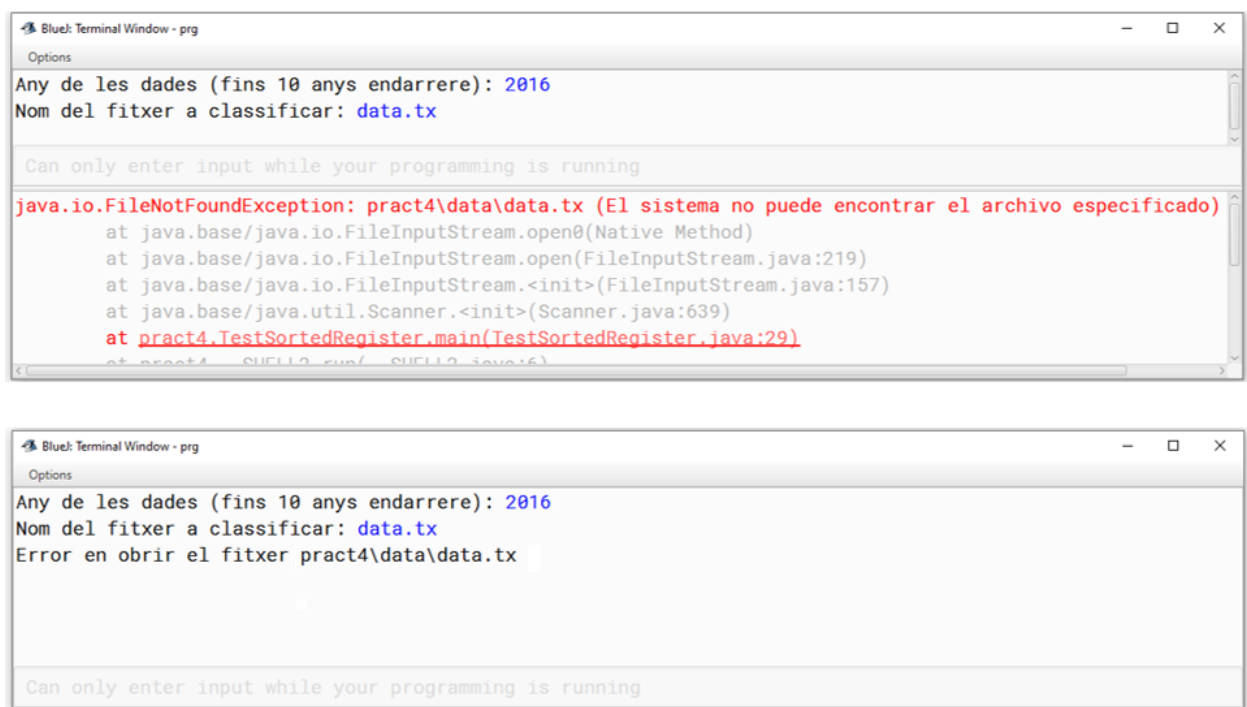


Figura 10: Gestió de l'excepció **FileNotFoundException**: l'execució de dalt és prèvia a completar el **main** de **TestSortedRegister** segons l'activitat 7, i la de baix és posterior.