Pràctica 4 PRG (2019/2020) OnLine

DURACIÓ 2 SESSIONS

Tractament d'Excepcions i Fitxers:

Tota la pràctica serà Online, però durant els horaris de classe haurà oberta una sessió de TEAMS per a dubtes i consultes.

NO CAL connectar-se a escriptori virtual però es possible fer-ho.

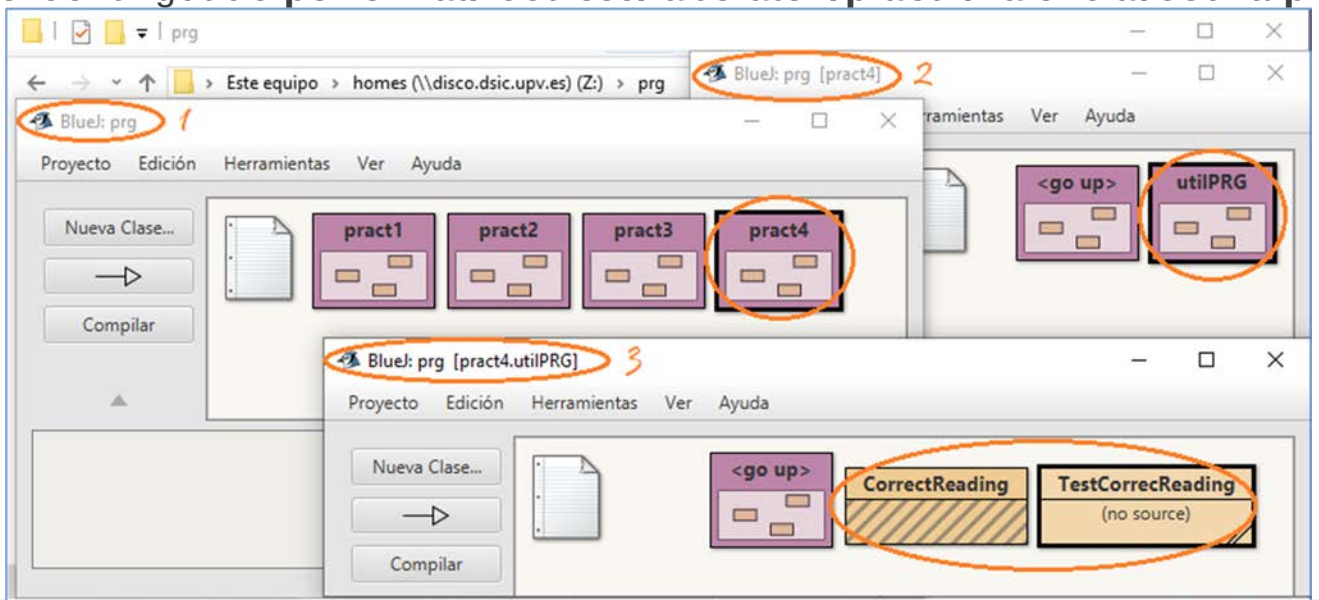
Sessió 1:

Recomanació Prèvia si ho fas a casa: reprodueix l'estructura de paquets al teu ordinador o connecta una VPN a la UPV i configura una unitat de disc en xarxa: **fileserver.dsic.upv.es/homes** amb el teu usuari del DSIC, i tindràs accés al teu directori del laboratori.

- Llistat d'activitats:
  - Activitat 1: preparar amb **BlueJ** el paquet **pract4**: dintre del projecte **prg** (al teu ordinador o al del lab)
  - Activitat 2: provar i examinar **`nextInt(Scanner, String)`**
  - Activitat 3: completar **`nextDoublePositive(Scanner, String)`**
  - Activitat 4: completar **`nextInt(Scanner, String, int, int)`**

Sessió 1:

- Llistat d'activitats:
  - Recomanació Prèvia si ho fas a casa: reprodueix l'estructura de paquets al teu ordinador o connecta una VPN a la UPV i configura una unitat de disc en xarxa: **fileserver.dsic.upv.es/homes** amb el teu usuari del DSIC, i tindràs accés al teu directori del laboratori.
  - Activitat 1: preparar amb **BlueJ** el paquet **pract4**: dintre del projecte **prg** (al teu ordinador o al del laboratori virtual)
    1. Crea un paquet dintre de **prg** anomenat **pract4** i
    2. dintre d'aquest altre anomenat **utilPRG** i tanca **BlueJ**
    3. Descarrega el contingut de **poliformat/recursos/laboratori/pract4/valencià/codi** a **prg/pract4/utilPRG**
    4. Obri **BlueJ**



Sessió 1:

- Llistat d'activitats(2):
  - Activitat 2: provar i examinar **`nextInt(Scanner, String)`**

1. Després de compilar, pots provar-ho en la *code pad*, però abans has de escriure les instruccions següent:

```
Import java.util.Scanner;
Scanner t = new Scanner(System.in);
intValor = CorrectReading.nextInt(t, "Valor: ");
```

2. Llig la documentació i mira el codi del mètode i prova els casos que llancen excepció

```
// ACTIVITAT 2:
/** Llig un valor des d'un Scanner i retorna el primer valor de tipus int llegit. <br><br>
 * Si el valor llegit no es un numero enter, mostra per pantalla el missatge: "Per favor, introdueix un numero enter correcte! ..."
 * <br><br>  La lectura es repeteix fins que el token llegit siga correcte, tornant el primer que siga enter.
 *
 * @param sc Scanner des d'on es fa la lectura.
 * @param msg String per preguntar a l'usuari sobre el valor.
 * @return int, valor enter.                                          */
public static int nextInt(Scanner sc, String msg) {
    int value = 0;
    boolean someError = true;
    do {
        try { System.out.print(msg);    value = sc.nextInt();    someError = false; }
        catch (InputMismatchException e) { System.out.println("Per favor, introdueix un numero enter correcte! ..."); }
        finally { sc.nextLine(); }
    } while (someError);
    return value;
}
```

Sessió 1:

- Llistat d'activitats(3):
  - Activitat 3: completar **`nextDoublePositive(Scanner, String)`**
    1. Modifica el mètode per a que capture l'excepció **`InputMismatchException`** igual que el mètode anterior i escriga un missatge apropiat. En la documentació tens exemples dels missatges. Has d'afegir el bloc **`try-catch-finally`** necessari.
    2. En acabant comprova el funcionament del teu mètode igual que has provat **`nextInt(Scanner, String)`**

```
// ACTIVITAT 3:
/** Llig un valor des d'un Scanner i retorna el primer valor de tipus double no negatiu llegit. <br><br>
 * Si el valor llegit es un numero real negatiu, mostra per pantalla el missatge: "Per favor, introdueix un valor correcte! ..."
 * <br><br>  Si el valor llegit no es un double, mostra per pantalla:  "Per favor, introdueix un numero real correcte! ..."
 * <br><br> La lectura es repeteix fins que siga correcte, tornant el primer que siga >= 0.0.
 *
 * @param sc Scanner des d'on es fa la lectura.
 * @param msg String per preguntar a l'usuari sobre el valor.
 * @return double, valor double no negatiu.                                    */
public static double nextDoublePositive(Scanner sc, String msg) {
    double value = 0.0;
    boolean someError = true;
    do {// COMPLETAR
        System.out.print(msg);    value = sc.nextDouble();
        if (value < 0.0) { System.out.println("Per favor, introdueix un valor correcte! ..."); }
        else { someError = false; }
        // COMPLETAR
        sc.nextLine();
        // COMPLETAR
    } while (someError);
    return value;
}
```

Sessió 1:

- Llistat d'activitats(4):
  - Activitat 4: completar **`nextInt(Scanner, String, int, int)`**
    1. Modifica el mètode per a que capture l'excepció **`InputMismatchException`** igual que el mètode anterior i escriga un missatge apropiat. A més, el mètode només ha de admetre valors entre els dos *integers* donats, escrivint un missatge si no es vàlid. En lloc de fer-ho mitjançant un **`if`**, has de fer-ho com diu la documentació del mètode, es a dir, llançant una excepció i tornant a capturar-la en el **`catch`**. En la documentació tens exemples dels missatges. Has d'afegir el bloc **`try-catch-finally`** necessari. I recorda que també tindràs que fer un **`throw`** i un **`new`** de la nova excepció.
    2. En acabant comprova el funcionament del teu mètode igual que has provat **`nextDoublePositive(Scanner, String)`**

nextInt

```
public static int nextInt(java.util.Scanner sc,
                          java.lang.String msg,
                          int lowerBound,
                          int upperBound)
```

Llig un valor des d'un Scanner i retorna el primer valor de tipus int llegit en el rang [lowerBound .. upperBound] on
Integer.MIN_VALUE <= lowerBound i upperBound <= Integer.MAX_VALUE.
- Si l'enter llegit esta fora del rang, llanca una excepcio de tipus IllegalArgumentException amb el missatge:
  "v no esta en el rang [lowerBound .. upperBound]" on v es el valor llegit i lowerBound, upperBound son els parametres.
A continuacio, captura aquesta excepcio i mostra per pantalla el missatge de l'excepcio junt amb el text: ". Per favor, introdueix un valor correcte! ..."
- Si el valor no es un enter, mostra per pantalla el missatge: "Per favor, introdueix un numero enter correcte! ..."
La lectura es repeteix fins que el token llegit siga correcte, tornant el primer que siga un enter valid.

Parameters:
sc - Scanner des d'on es fa la lectura.
msg - String per preguntar a l'usuari sobre el valor.
lowerBound - int limit inferior.
upperBound - int limit superior.

Returns:
int, valor enter dins dels limits.

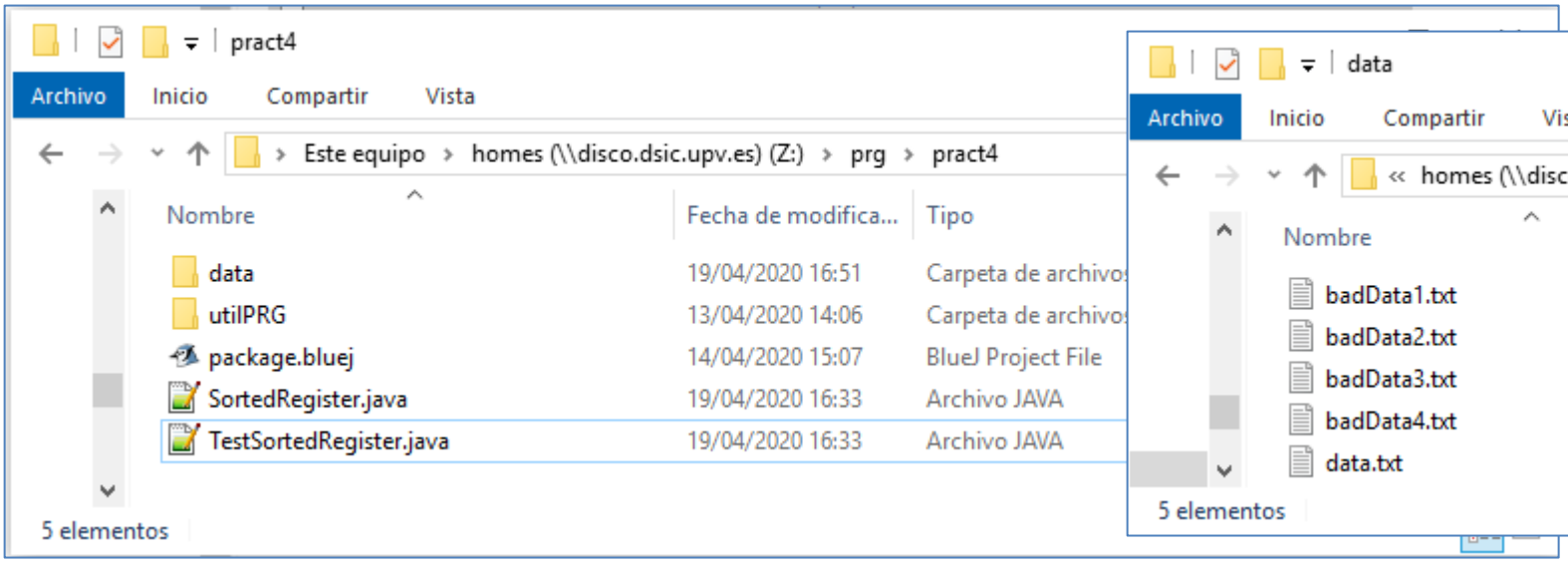
Sessió 2:

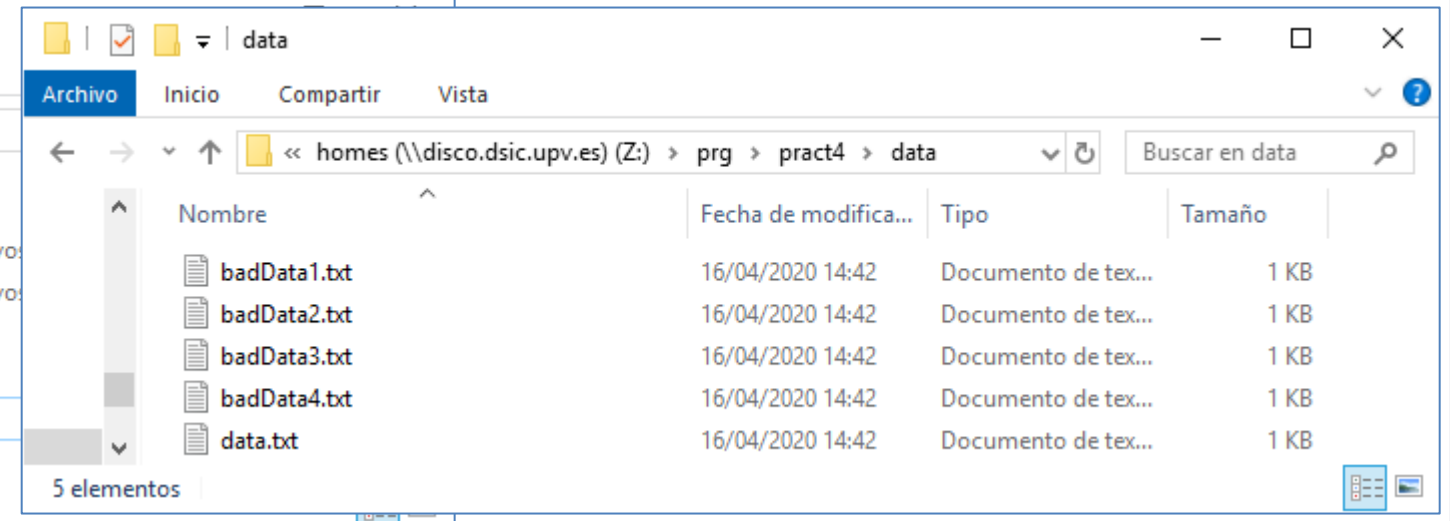
Per a fer aquesta segona sessió has de haver acabat les activitats de la primera, perquè els mètodes implementats es puguen utilitzar d'ací endavant.

- Llistat d'activitats:
  - Activitat 1: instal·lació de les classes i fitxers de prova.
  - Activitat 2: examen i prova de la classe **`SortedRegister`**.
  - Activitat 3: examen i prova del mètode **`handleLine(String)`**.
  - Activitat 4: finalització del mètode **`handleLine(String)`**.
  - Activitat 5: captura d'excepcions en el mètode **`add(Scanner)`**.
  - Activitat 6: desenvolupament del mètode **`add(Scanner, PrintWriter).`**
  - Activitat 7: captura d'excepcions de classe **`FileNotFoundException`** en el programa **`TestSortedRegister`**.

Sessió 2:

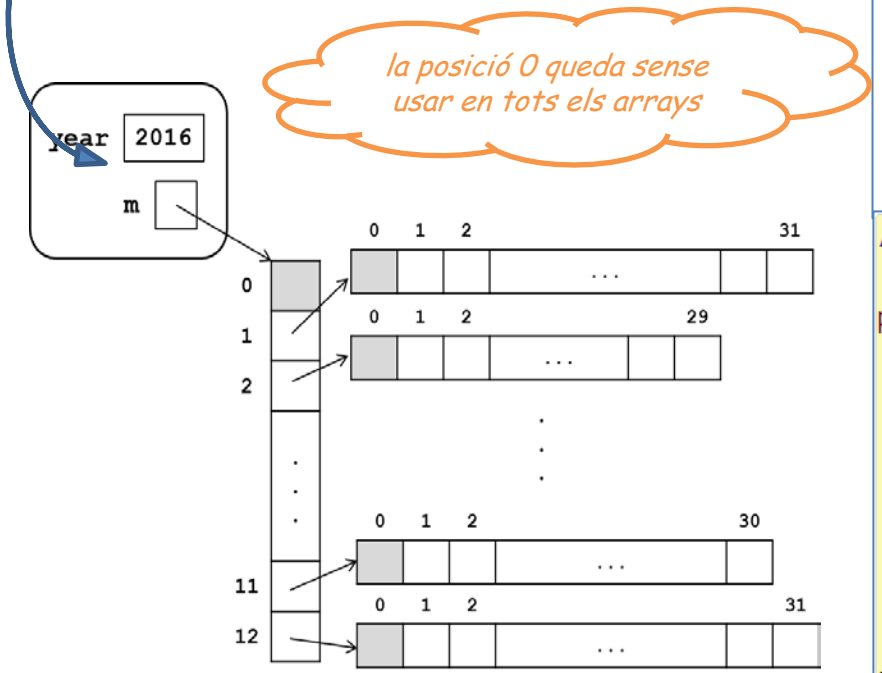
- Llistat d'activitats (1):
  - Activitat 1: preparar amb **BlueJ** el paquet **pract4**: dintre del projecte **prg** (al teu ordinador o al del laboratori virtual)
    - Descarregar de **PoliformaT** de la carpeta de material de la pràctica 4, els fitxers **`SortedRegister.java`** i **`TestSortedRegister.java`** 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**.





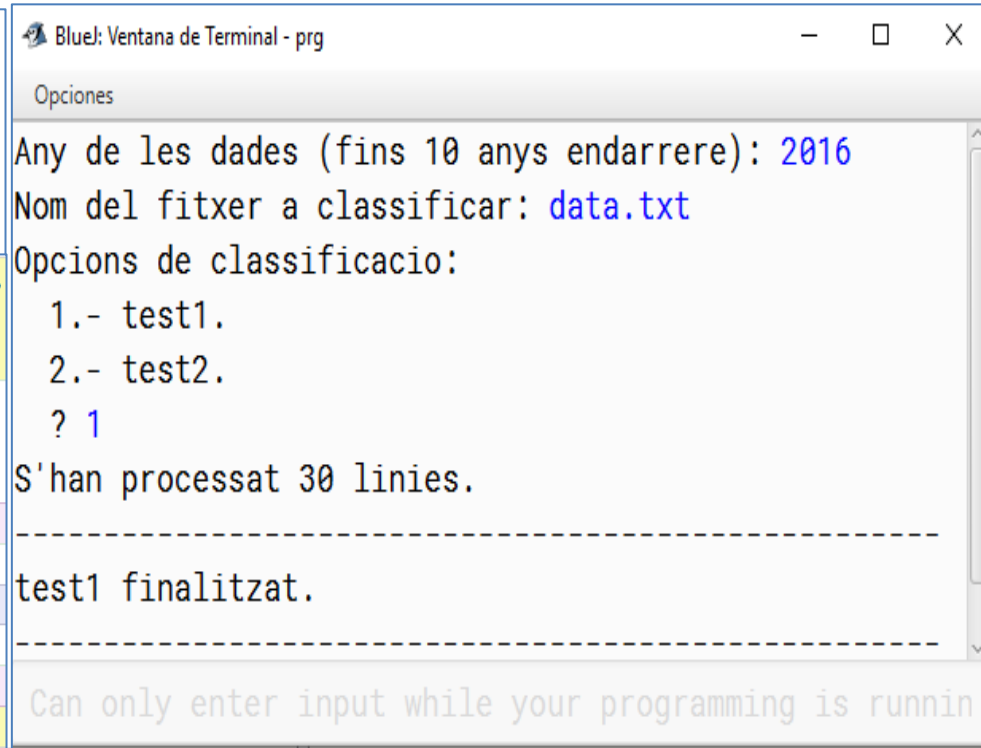
Sessió 2:

- Llistat d'activitats (2):
  - Activitat 2: examen i prova de la classe **`SortedRegister.java`**
    - Examina el contingut, atributs i mètodes de la classe **`SortedRegister.java`** i més concretament:
      - La matriu de dades **`private int[][] m`** i de pas pega una ullada a la resta de atributs auxiliars definits.
      - El mètodes **`int add(Scanner)`**. A aquest mètode li falta tractar les excepcions (mes endavant en altra activitat).
      - El mètodes **`handleLine(String)`**. A aquest mètode li falta llançar una excepció en un cas (mes enfavant en altra activitat).
      - 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) i mira els resultats.



`m` es un <u>array de arrays</u> de enters, això vol dir que el primer claudàtor representa els mesos de l'any (a comptar des de 0) i el segon claudàtor els dia del mes (a comptar des de 0) per a això tenim aquest constructor:

```
/** Crea un SortedRegister per a l'any dataYear.
 *  @param dataYear int.                          */
public SortedRegister(int dataYear) {
    this.year = dataYear;
    boolean leap = isLeap(this.year);
    this.m = new int[13][];
    for (int month = 1; month <= 12; month++) {
        int numD = DAYS[month];
        if (month == 2 && leap) { numD++; }
        this.m[month] = new int[numD + 1];
    }
}
```



Sessió 2:

- Llistat d'activitats(3):
  - Activitat 3: examen i prova del mètode **`handleLine(String)`**

    El mètode detecta els tres següents erros i llança les excepcions descrites:

    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 línia 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 vàlids d'una component de la matriu **`m`**, es crea i es llança una excepció **`IllegalArgumentException`** amb el missatge "**Data incorrecta**.".

Llig la documentació i mira el codi del mètode i prova els casos que llancen excepció

Tornar a executar el main de **`TestSortedRegister`**, opció **1** any **2016** i fitxer **badData1.txt**

Mira **badData1.txt** i dedueix quina excepció es llança a la primera línia llegida.

Per tant el fitxer **result.txt** no haurà pogut omplir-se... Comprova el seu contingut.

```
 *  @throws NumberFormatExceptionException si es llig una dada no entera.
 *  @throws IllegalArgumentException si la linia no conte 3 dades,
 *  o no s'ajusten al format de data i quantitat correctes.
 */
private void handleLine(String line) {
    String[] data = line.trim().split("[ ]+");
    // data es un array amb tantes components com tokens apareixen en line
    if (data.length != 3) {
        throw new IllegalArgumentException("La linia no conte tres dades.");
    }
    int day = Integer.parseInt(data[0]); // Pot produir NumberFormatException
    int month = Integer.parseInt(data[1]); // Pot produir NumberFormatException
    int amount = Integer.parseInt(data[2]); // Pot produir NumberFormatException
    if (month <= 0 || month >= this.m.length || day <= 0 || day >= this.m[month].length) {
        throw new IllegalArgumentException("Data incorrecta.");
    }
    // COMPLETAR
    this.m[month][day] += amount; // Per la comprovacio anterior, no es pot produir
                                  // ArrayIndexOutOfBoundsException
}
```

Sessió 2:

- Llistat d'activitats(4):
  - Activitat 4: finalització del mètode **`handleLine(String)`**.

- El mètode està incomplet, perquè si la quantitat de accidents es negativa no te sentit i el mètode no hauria de fer l'ultima línia.
- Afegir en el mètode una instrucció que comprove si la quantitat (**`amount`**) és negativa, en aqueix cas cree i llance una excepció **`IllegalArgumentException`** amb el missatge "**Quantitat negativa.**".
- Executar el **`main`** de la classe **`TestSortedRegister`**, opció **1,** any **2016** i fitxer **data1.txt**
- Comprova que, quan es processe la línia amb la quantitat negativa, es produïsca l'excepció corresponent.

```
 *  @throws NumberFormatExceptionException si es llig una dada no entera.
 *  @throws IllegalArgumentException si la linia no conte 3 dades,
 *  o no s'ajusten al format de data i quantitat correctes.
 */
private void handleLine(String line) {
    String[] data = line.trim().split("[ ]+");
    // data es un array amb tantes components com tokens apareixen en line
    if (data.length != 3) {
        throw new IllegalArgumentException("La linia no conte tres dades.");
    }
    int day = Integer.parseInt(data[0]); // Pot produir NumberFormatException
    int month = Integer.parseInt(data[1]); // Pot produir NumberFormatException
    int amount = Integer.parseInt(data[2]); // Pot produir NumberFormatException
    if (month <= 0 || month >= this.m.length || day <= 0 || day >= this.m[month].length) {
        throw new IllegalArgumentException("Data incorrecta.");
    }
    // COMPLETAR
    this.m[month][day] += amount; // Per la comprovacio anterior, no es pot produir
                                  // ArrayIndexOutOfBoundsException
}
```

Sessió 2:

- Llistat d'activitats(5):
  - Activitat 5: captura d'excepcions en el mètode **`add(Scanner)`**.
    1. **Modifica** el mètode per a que capture les excepcions que es produeixen en **`handleLine(String)`** i que ara son només propagades. Ara, quan es troba una línia incorrecta es deixa de tractar el **`Scanner`** d'entrada i se fa el **`return`**.
    2. Volem que es capturen les excepcions apropiades i s'escriga un missatge per error, sense parar la execució.

```
public int add(Scanner sc) {
    // A COMPLETAR AMB LA CAPTURA D'EXCEPCIONS
    int count = 0;
    while (sc.hasNext()) {
        count++;
        this.handleLine(sc.nextLine());
    }
    return count;
}
```

AFEGIR

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

3. Després prova el funcionament del **`add`** modificat:

Casos de prova del **`main`** de la classe **`TestSortedRegister`**, amb opció **1:**

- 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`.

Sessió 2:

- Llistat d'activitats(6):
  - Activitat 6: desenvolupament del mètode **`add(Scanner, PrintWriter).`**
    1. Escriu el cos **D'UN ALTRE MÈTODE** que també s'anomena **`add`** però que té dos paràmetres, el primer un **`Scanner`** com el **`add`** original i el segon un **`PrintWriter`** que servisca per a bolcar la informació de les línies errònies a un fitxer
    2. Aquest nou mètode el trobaràs en el codi de la classe **`SortedRegister.java`** baix dels mètodes **`handleLine(String)`** i de **`save(PrintWriter)`**.
    3. **NO HAS DE MODIFICAR L'ALTRE MÈTODE `add(Scanner)` NECESSITES ELS DOS PER A FINALITZAR LA PRÀCTICA**

IMPORTANT

```
 *  Classifica ordenadament les dades llegides del Scanner s. Es filtren les dades que tingueren
 * algun defecte de format, emetent un informe d'errors.
 *   Precondicio: El format de linia recognoscible es     "dia mes quantitat"
 * on dia i mes han de ser enters corresponents a una data valida, i quantitat ha de ser un enter > 0.
 *   La quantitat llegida s'acumula en el registre que es porta per al dia del mes.
 * En err s'escriuen les linies defectuoses, indicant el nombre de linia.
 *  @param sc Scanner font de les dades.
 *  @param err Printwriter desti de l'informe d'errors.
 *  @return int, el nombre de linies processades.
public int add(Scanner sc, PrintWriter err) {
    int count = 0;
    // COMPLETAR
    return count;
}
```

4. Per a provar aquest mètode cal completar el **`test2`** de la classe **`TestSortedRegister.java:`**
5. **NO MODIFIQUEU EL `test1`,** però si que podeu tomar-ho com a model però cridant al **`add(Scanner, PrintWriter)`** que heu completat en lloc de a l'altre després podeu fer les proves triant opció **2** en lloc d'opció **1**

Sessió 2:

- Llistat d'activitats(7):
  - Activitat 7: captura d'excepcions de classe **`FileNotFoundException`** en el programa **`TestSortedRegister`**
    1. Fixat en la capçalera del mètode **`main`** de la classe **`TestSortedRegister`** i veuràs que propaga **`FileNotFoundException`** i que no la tracta en cap lloc. Això significa que si un fitxer no existeix, el programa termina abruptament, perquè es el tractament per defecte de la màquina virtual.

```
/**
 * Classe TestSortedRegister. Test de la classe SortedRegister.
 *
 * @author (PRG. ETSINF - UPV)
 * @version (2019/20)
 */
public class TestSortedRegister {
    /** No hi ha objectes d'aquesta classe. */
    private TestSortedRegister() { }

    public static void main(String[] args) throws FileNotFoundException {
```

    2. Modifica el **`main`** (i la seua capçalera) per a capturar i tractar aquesta excepció. Per a poder fer-ho pots aprofitar i afegir aquest codi que pots trobar al butlletí.
    3. I en acabant prova tota la pràctica i els diferents casos.

```
...
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(); }
}
```