



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica



Tema 4. Entrada i eixida. Fluxes i fitxers

Programació (PRG)

Curs 2019/20

Departament de Sistemes Informàtics i Computació



Continguts

Duració: 1 sessió

1. Introducció
 - Fluxes: `InputStream`, `OutputStream`
 - Fitxers
2. Accés al sistema d'arxius
 - La classe `File`
3. Fitxers de text
 - Escriptura: la classe `PrintWriter`
 - Lectura: la classe `Scanner`
4. Tancament de fitxers
5. Fitxers binaris. Fitxers binaris d'accés seqüencial
6. Resum i recomanacions al treball amb fitxers
7. Jerarquia d'excepcions en entrada/eixida

Encara que en aquest document estan les transpescades d'aquest apartat, NO va per a examen.
Més informació sobre aquest contingut a les seccions 15.3, 15.4 i 15.5 del Capítol 15 del llibre de l'assignatura

- Pràctiques relacionades:
 - PL 4. Tractament d'excepcions i fitxers (3 sessions)

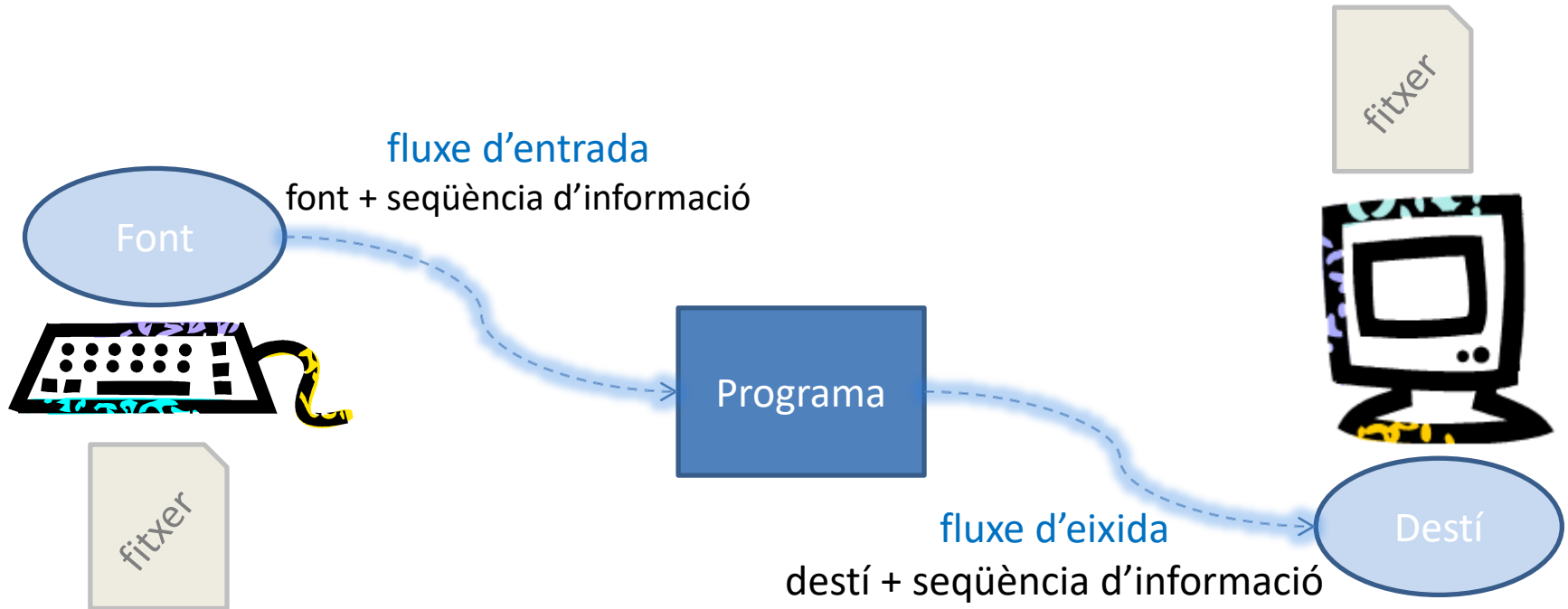


- **Descarrega** (del Tema 4 de PoliformaT) els fitxers ***exemplesT4.jar*** i ***exercicisT4.jar*** en una carpeta ***PRG/Tema 4*** dins del teu ***disc W***
- Des de l'opció **Projecte** de **BlueJ**, usa l'opció **Open ZIP/JAR...** per tal d'obrir-los com projectes **BlueJ** que contenen diferents paquets i prepara't per usar-los



Introducció: fluxes

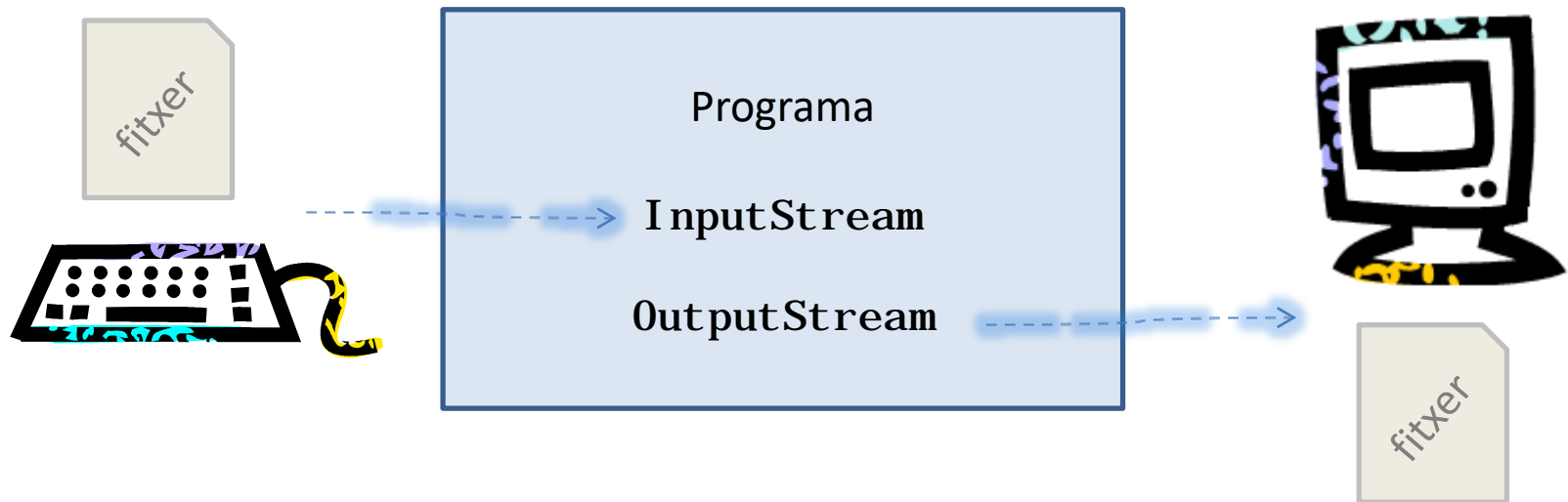
- En Java l'entrada/eixida es realitza utilitzant fluxes (*streams*), que són seqüències d'informació (seqüències de bytes) que tenen una font (*fluxes d'entrada*) o un destí (*fluxes d'eixida*).



- Associats a tot programa Java en execució, hi ha dos fluxes especials, un d'entrada o *entrada estàndard* (`System.in`), i un d'eixida o *eixida estàndard* (`System.out`). Estan associats, per defecte, al teclat i a la pantalla, respectivament.

Introducció: fluxes

- Els fluxes d'entrada en Java són de la classe `InputStream` (i les seues derivades com `FileInputStream`: fluxes on la font és un fitxer).
- Els fluxes d'eixida en Java són de la classe `OutputStream` (i les seues derivades com `FileOutputStream`: fluxes on el destí és un fitxer).



Introducció: fitxers

- Un fitxer és un conjunt de bits guardat en un dispositiu secundari d'emmagatzemament (disc dur, USB stick, etc.).
 - Permet emmagatzemar les dades existents en memòria d'un programa per a ser utilitzades posteriorment (per aquest o un altre programa).
- Característiques principals :
 - Nom (p.e. `fi t x e r . t x t`).
 - Ruta en el dispositiu d'emmagatzemament (p.e. `/home/lucas/docs/fi l e . t x t`).
 - Amplària, típicament expressada en bytes (Kbytes, Mbytes, Gbytes, etc.).
- Característiques addicionals:
 - Permisos d'accés (dependents del sistema de fitxers).
 - Data de la darrera modificació.
 - ...
- Accions principals sobre fitxers:
 - Obrir, Llegir, Tancar.
 - Obrir, Escriure, Tancar.

Introducció: tipus de fitxers

- En general, distingim entre dos tipus de fitxers:

Fitxers de text

- Seqüència de caràcters.
- Interpretable per un ésser humà.
- Generalment portable (llegible en diferents equips).
- Escriptura/lectura menys eficient que per a fitxers binaris.
- Requereix més grandària que un fitxer binari per representar la mateixa informació.
 - Ex. Un enter de 10 dígitos en un fitxer de text ocupa 10 bytes (assumint codificació ASCII d'1 byte/caràcter).

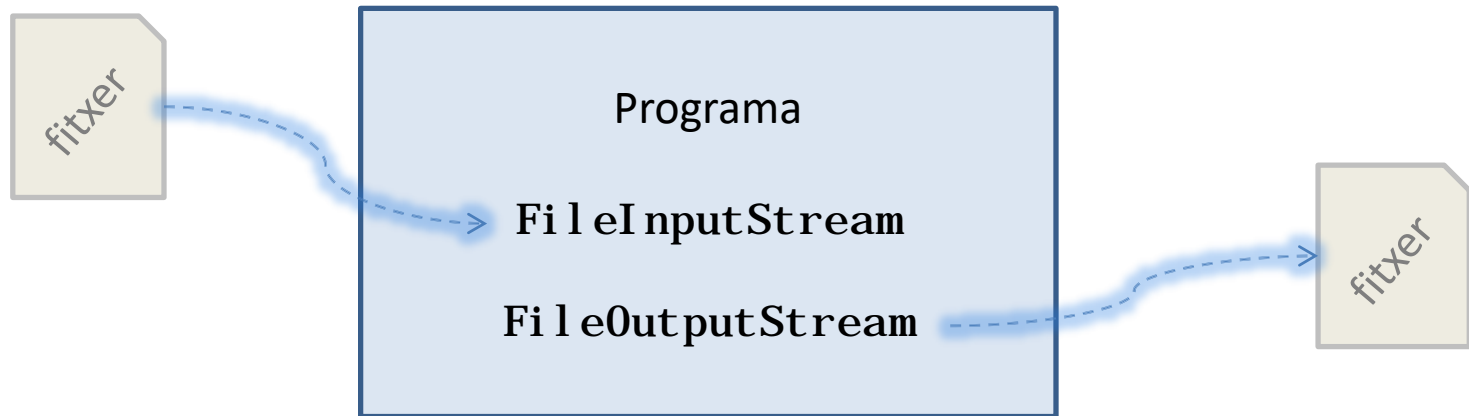
Fitxers binaris

- Seqüència de bytes interpretables com a valors (primitius o objectes).
- No interpretable per un ésser humà.
- Generalment no portable (ha de ser llegit en el mateix tipus d'ordinador i amb el mateix llenguatge de programació en que va ser escrit).
- Escriptura/lectura eficient.
- Emmagatzematge eficient de la informació.
 - Ex. Un enter de 10 dígitos en un fitxer binari ocupa 4 bytes.

- En Java, els fitxers binaris SÍ són independents de la plataforma (poden ser llegits en diferents plataformes), això sí, des de Java.

Introducció: fitxers

- En general:
 - Llegir d'un fitxer en Java requereix definir un fluxe d'entrada, la font del qual siga el fitxer (un `FileInputStream`)
 - Escriure en un fitxer en Java requereix definir un fluxe d'eixida, el destí del qual siga el fitxer (un `FileOutputStream`)



- Llegir i escriure directament d'aquests fluxes és farragós (els seus mètodes lligen i escriuen byte a byte).
- Java proporciona altres classes que faciliten la gestió d'aquests fluxes, especialitzades al tipus de fitxers (de text, binaris).

Funcionalitats dels fitxers en Java

- Java permet:
 1. Interactuar amb el sistema d'arxius independentment del seu tipus (FAT32, NTFS, EXT3, etc.) i del S.O. (Windows, Linux, MAC, etc.).
 2. Crear i consumir fitxers de text (compostos tant per les representacions en caràcters dels tipus primitius com per `Strings`).
 3. Produir i llegir fitxers binaris compostos per dades de tipus primitius i `Strings`.
 4. Treballar amb fitxers binaris formats per objectes (serialització d'objectes).
 5. Manipular fitxers binaris a baix nivell (Entrada/Eixida a nivell de bytes).
- En aquest tema ens centrarem en les característiques dels Punts 1 al 3.

Accés al sistema d'arxius: `File`

- La classe `File` (del paquet `java.io`) permet representar en Java les rutes dels fitxers i directoris del sistema.

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/File.html>

Mètode / Constructor	Descripció
<code>File(String pathname)</code>	Crea un nou objecte de tipus <code>File</code> a partir de la seua ruta
<code>boolean delete()</code>	Esborra el fitxer/directori
<code>boolean exists()</code>	Indica si el fitxer/directori existeix
<code>String getAbsolutePath()</code>	Retorna la ruta absoluta del fitxer/directori (nom des de l'arrel)
<code>String getName()</code>	Retorna el nom del fitxer/directori (sense la ruta)
<code>String getParent()</code>	Retorna la ruta al fitxer/directori (sense el nom)
<code>long length()</code>	Retorna la llargària del fitxer. No vàlid per a directoris
<code>File[] listFiles()</code>	Torna un array amb la llista de fitxers al directori
<code>boolean isDirectory()</code>	Indica si es tracta d'un directori
<code>boolean isFile()</code>	Indica si es tracta d'un fitxer
...	



Crear un objecte `File` no modifica el sistema d'arxius; només es produeixen canvis si s'invoquen certs mètodes de la classe (per exemple, `delete` amb èxit, `mkdir`, etc.).

Accés al sistema d'arxius: File

- Exemple d'ús de la classe File per tractar amb el sistema d'arxius



BlueJ: exemplesT4

```
import java.io.File;
public class TestFile {
    private TestFile() { }

    public static void main(String[] args) {
        File f = new File("/home/pcasals/file.txt");
        System.out.println("getName(): " + f.getName());
        System.out.println("getParent(): " + f.getParent());
        if (f.exists()) {
            System.out.println("El fitxer existeix!");
            System.out.println("length(): " + f.length());
        }
        else { System.err.println("El fitxer NO existeix!"); }
    }
}
```

- f és un **descriptor** de fitxer, descriu una ruta en el sistema d'arxius
- getName() obté "file.txt"
- getParent() obté "/home/pcasals"

Exercici: classe InfoFile

- Completa el codi del `main` de la classe `InfoFile` al paquet `fitxersText` del projecte *BlueJ exercicisT4*



BlueJ: exercicisT4 [fitxersText]

```
import java.io.File;

/**
 * Mostra informacio basica sobre un fitxer (nom, directori i tamany en kbytes).
 * S'executa invocant main amb un parametre que es la ruta relativa del fitxer.
 * Per exemple, el nom relatiu d'aquest fitxer es fitxersText/InfoFile.java.
 * @author PRG
 * @version Curs 2019/20
 */
public class InfoFile {
    private InfoFile() { }

    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("Especifica el nom d'un fitxer");
        }
        else {
            String filePath = args[0];
            File f = /* COMPLETAR - crear el File de filePath */;
            if (f.isFile()) {
                System.out.println("Nom del fitxer: " + /* COMPLETAR */);
                System.out.println("Directori: " + /* COMPLETAR */);
                System.out.println("Tamany (kbytes): " + /* COMPLETAR */);
                System.out.println("Ruta absoluta: " + /* COMPLETAR */);
            }
        }
    }
}
```

Fitxers de text. Escriitura: `PrintWriter`

- La forma més còmoda de generar i escriure fitxers de text és mitjançant la classe `PrintWriter` (del paquet `java.io`):
 - Proporciona mètodes que faciliten l'escriitura com text de valors de tipus primitius i `Strings`.

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/PrintWriter.html>

Mètode / Constructor	Descripció
<code>PrintWriter(File file)</code>	Crea un nou <code>PrintWriter</code> a partir d'un <code>File</code> . Pot llançar l'excepció <code>FileNotFoundException</code>
<code>PrintWriter(String filename)</code>	Crea un nou <code>PrintWriter</code> a partir del nom d'un fitxer. Pot llançar l'excepció <code>FileNotFoundException</code>
<code>PrintWriter(OutputStream out)</code>	Crea un nou <code>PrintWriter</code> a partir d'un stream
<code>void print(float f)</code>	Escriu un <code>float</code> (sense canvi de línia)
<code>void println(float f)</code>	Escriu un <code>float</code> (i afegeix un canvi de línia)
<code>void print(boolean b)</code>	Escriu un <code>boolean</code> (sense canvi de línia)
...	...
<code>PrintWriter printf(Locale l, String format, Object... args)</code>	Escriu un <code>String</code> formatejat utilitzant el format i els arguments especificats
<code>boolean checkError()</code>	Comprova si hi ha hagut error en l'escriitura anterior
<code>void close()</code>	Tanca el <code>PrintWriter</code>

Fitxers de text. Escriptura: `PrintWriter`

- Exemple de creació d'un `PrintWriter`:

```
PrintWriter pw = new PrintWriter(new File("/tmp/f.txt"));
```

- Al crear un `PrintWriter`:
 - Si el fitxer no existeix es crea un de nou, si existeix es trunca el seu tamany a zero.
 - Si ocorre algun error (el fitxer no és accessible en el sistema d'arxius, ...) es llança l'excepció *checked* `FileNotFoundException` (que haurà de ser tractada).
- Quan s'ha creat amb èxit un `PrintWriter` i ja no s'ha d'usar més en l'aplicació, es pot tancar amb el mètode `close()`.

Exemple d'ús de PrintWriter

- Exemple de programa que usa PrintWriter:



BlueJ: exemplesT4

```
import java.io. File;
import java.io. FileNotFoundException;
import java.io. PrintWriter;
public class TestPrintWriter {
    private TestPrintWriter() { }
    public static void main(String[] args) {
        String fitxer = "fileTPW.txt";
        try {
            PrintWriter pw = new PrintWriter(new File(fitxer));
            pw.print("Setze jutges d'un jutjat ");
            pw.println("mengen fetge d'un penjat ");
            pw.println(4.815162342);
            pw.close();
        } catch (FileNotFoundException e) {
            System.err.println("Problemes en obrir el fitxer " + fitxer);
        }
    }
}
```

- El programa escriu en el fitxer:
Setze jutges d'un jutjat mengen fetge d'un penjat
4.815162342

Fitxers de text. Lectura: Scanner

- La classe `Scanner` (del paquet `java.io`) facilita la lectura desde diversos tipus de font.
<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Scanner.html>
- Els mètodes constructors permeten crear un `Scanner` la font de caràcters del qual siga de diferents classes:

Constructor	Descripció
<code>Scanner(File source)</code>	Crea un nou <code>Scanner</code> a partir d'un <code>File</code> . Si el fitxer no existeix o no es pot obrir, llança <code>FileNotFoundException</code>
<code>Scanner(String src)</code>	Crea un nou <code>Scanner</code> per a llegir dades de l' <code>String src</code>
<code>Scanner(InputStream source)</code>	Crea un nou <code>Scanner</code> a partir d'un <code>InputStream</code> (l'entrada estàndard <code>System.in</code> és un <code>InputStream</code>)
...	...

- Conté altres mètodes per configurar l'`Scanner`, tancar-lo, etc.

Mètode	Descripció
...	...
<code>Scanner useLocale(Locale l)</code>	Estableix la configuració local del <code>Scanner</code> a la configuració especificada pel <code>Locale l</code>
<code>void close()</code>	Tanca l' <code>Scanner</code> , alliberant els recursos associats a ell

Fitxers de text. Lectura: Scanner

- Exemples de creació d'un Scanner:

```
Scanner teclat = new Scanner(System.in).useLocale(Locale.US);
```

```
Scanner text = new Scanner("Text a escanejar. \n En 2 línies");
```

```
Scanner fitxer = new Scanner(new File("/tmp/fEntrada.txt"));
```

- Si el fitxer especificat no està accessible en el sistema d'arxius, el constructor `Scanner(File)` llança l'excepció *checked* **FileNotFoundException** (que haurà de ser tractada).
- Quan s'ha creat amb èxit un Scanner i ja no s'ha d'usar més en l'aplicació, es pot tancar amb el mètode `close()`.

Fitxers de text. Lectura: Scanner

- Els principals mètodes que es poden aplicar a un Scanner són els de lectura.
- En un Scanner les dades de la font o entrada es consideren dividits en *tokens*:
 - paraules o seqüències de caràcters entre separadors (blancs Java per defecte) que es processen d'un en un.
- Conté mètodes que permeten traduir els tokens a valors de diversos tipus primitius:

Mètodes de lectura	Descripció
<code>boolean hasNext()</code>	Retorna <code>true</code> si queda almenys un token a l'entrada
...	...
<code>boolean hasNextInt()</code>	Retorna <code>true</code> si el següent token de l'entrada és un enter
<code>int nextInt()</code>	Retorna el següent token de l'entrada com un enter. Si no ho és, llança <code>InputMismatchException</code> . Si no queden tokens per llegir, llança <code>NoSuchElementException</code>
...	...
<code>String next()</code>	Retorna el següent token de l'entrada. Si no queden tokens per llegir, llança <code>NoSuchElementException</code>
<code>String nextLine()</code>	Retorna la resta de la línia. Si no queden línies per llegir, llança <code>NoSuchElementException</code>
...	...

... i els anàlegs per a `boolean`, `byte`, `short`, `long`, `float`, `double`

Fitxers de text. Lectura: Scanner

- Un Scanner conté un *buffer* de memòria (char[]) on disposa dels caràcters que li apleguen de la font i un indicador (posi ti on) d'on comença el següent token a llegir.

The screenshot displays the BlueJ IDE interface with the following components:

- scanner1 : Scanner** (Main object view):
 - Fields and values:
 - private CharBuffer buf (linked to buf : HeapCharBuffer)
 - private int position: 3
 - private Matcher matcher
 - private Pattern delimPattern
 - private Pattern hasNextPattern: null
 - private int hasNextPosition: 0
 - private String hasNextResult: null
 - private Readable source
 - private boolean sourceClosed: false
 - private boolean needInput: false
 - private boolean skipped: false
 - private int savedScannerPosition: -1
 - Buttons: Inspect, Show static fields
- buf : HeapCharBuffer** (Nested view):
 - Field: char[] hb (linked to hb : char[])
 - Buttons: Inspect
- hb : char[]** (Nested view):
 - Field: int length: 1024
 - Array elements (indices 0-7):
 - [0]: ''
 - [1]: 'u'
 - [2]: 'n'
 - [3]: ''
 - [4]: '1'
 - [5]: '\n'
 - [6]: 'd'
 - [7]: 'o'
 - Buttons: Inspect, Get, Show static fields, Close
- Code Editor**:

```
import java.util.Scanner;  
new Scanner(" un 1\ndos 2\ntres 3")  
<object reference> (Scanner)  
scanner1.next()  
"un" (String)
```
- Object Browser** (Bottom left):
 - scanner1: Scanner

A blue arrow points from the **position** field of the **scanner1** object to the **hb** array in the **hb : char[]** view, specifically pointing to the element at index 3.

Fitxers de text. Lectura: Scanner

- En el *CodePad* de [BlueJ](#), continua fent les següents lectures de l'Scanner i fixa't en com canvia el valor de `posi ti on` després de cada lectura i fins esgotar-se l'entrada.

```
import java.util.Scanner;

new Scanner(" un 1\ndos 2\ntres 3")
<object reference> (Scanner)
scanner1.next()
"un" (String)
scanner1.nextInt()
1 (int)
scanner1.next()
"dos" (String)
scanner1.nextLine()
" 2" (String)
scanner1.nextLine()
"tres 3" (String)
scanner1.nextLine()
Exception: java.util.NoSuchElementException (No line found)
```

scanner1 : Scanner

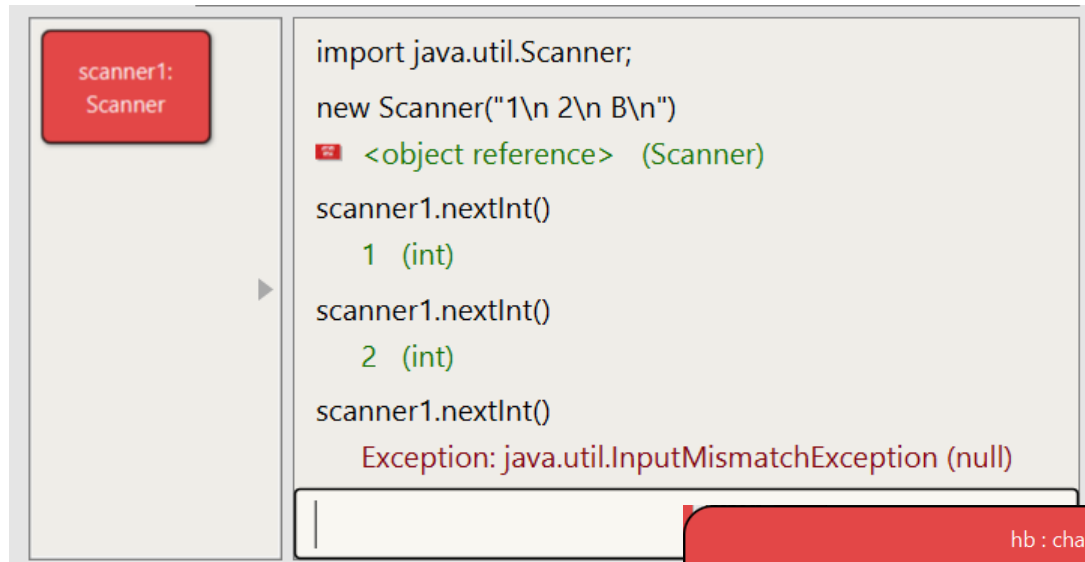
Fitxers de text. Lectura: Scanner

Excepcions

- Tots els mètodes de lectura com `next()`, `nextInt()`, `nextDouble()`, etc., llancen l'excepció (*unchecked*) **NoSuchElementException** si la font està esgotada.
- A més, els mètodes de lectura de tipus numèric (`nextInt()`, `nextDouble()`, etc.) o boolean (`nextBoolean()`), abans d'extraure el token de l'Scanner l'examinen per veure si s'ajusta al format corresponent (o compatible):
 - Si s'ajusta, calculen el valor corresponent i el retornen com resultat. Els caràcters del token es rebutgen (`position` “avança” fins el següent).
 - Si no s'ajusta, llancen l'excepció (*unchecked*) **InputMismatchException**, però els caràcters del token examinat no es rebutgen (`position` “no avança”): si es captura l'excepció, els caràcters segueixen estant a disposició per a la següent lectura.

Fitxers de text. Lectura: Scanner

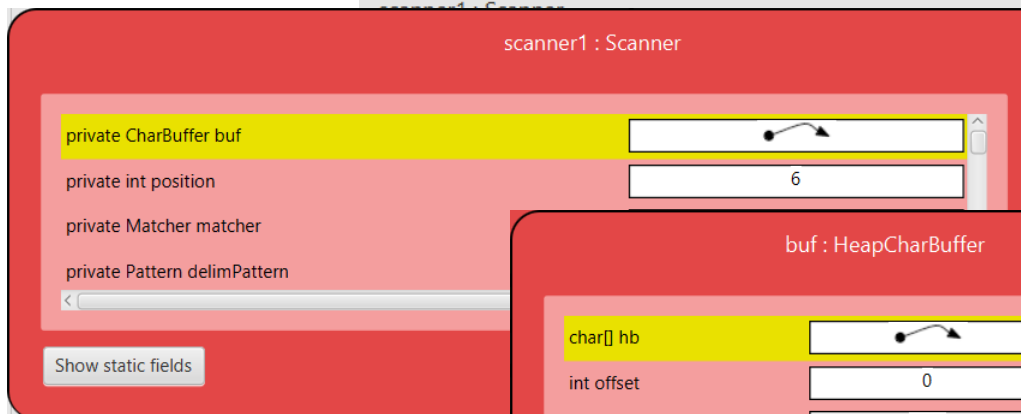
- Prova ara a executar les següents instruccions en el *CodePad* de [BlueJ](#), fixa't en el valor de `position` quan es produeix l'excepció `InputMismatchException`.



```
import java.util.Scanner;

new Scanner("1\n 2\n B\n")

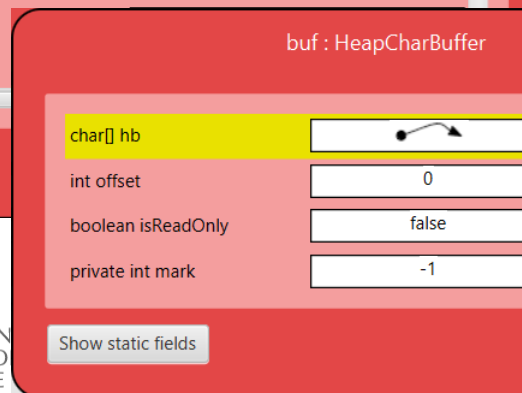
scanner1.nextInt()
    1 (int)
scanner1.nextInt()
    2 (int)
scanner1.nextInt()
Exception: java.util.InputMismatchException (null)
```



scanner1 : Scanner

private CharBuffer buf	
private int position	6
private Matcher matcher	
private Pattern delimPattern	

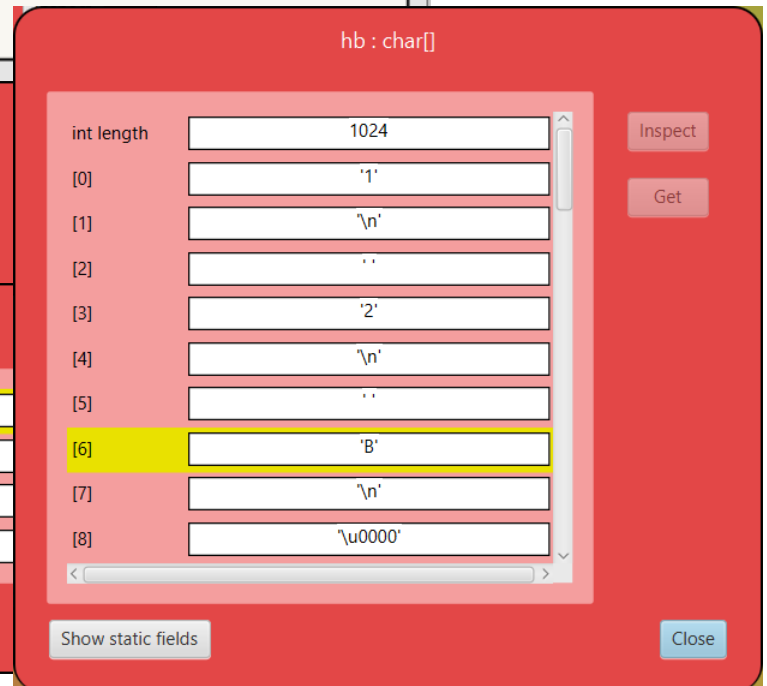
Show static fields



buf : HeapCharBuffer

char[] hb	
int offset	0
boolean isReadOnly	false
private int mark	-1

Show static fields



hb : char[]

int length	1024
[0]	'1'
[1]	'\n'
[2]	' '
[3]	'2'
[4]	'\n'
[5]	' '
[6]	'B'
[7]	'\n'
[8]	'\u0000'

Show static fields

Inspect

Get

Close

Exemple d'ús d'Scanner



BlueJ: exemplesT4

coses.txt

1 2

3 4

Multiplica't per zero!

```
import java.io. File;
import java.io. FileNotFoundException;
import java.util. InputMismatchException;
import java.util. Scanner;
public class TestScanner {
    private TestScanner() { }
    public static void main(String[] args) {
        System.out.println("Es lligen 3 números i una línia de text");
        Scanner scanner = null;
        try {
            scanner = new Scanner(new File("coses.txt"));
            int n1 = scanner.nextInt();
            int n2 = scanner.nextInt(), n3 = scanner.nextInt();
            scanner.nextLine();
            String linia = scanner.nextLine();
            System.out.println("Els tres números són: "
                + n1 + ", " + n2 + ", " + n3);
            System.out.println("La línia és: " + linia);
            scanner.close();
        } catch (FileNotFoundException ex) {
            System.err.println("El fitxer no existeix. \n" + ex.getMessage());
        } catch (InputMismatchException ex) {
            System.err.println("Error al llegir. " + ex.getMessage());
        }
    }
}
```

Exemple d'ús d'Scanner i de PrintWriter

- Programa que escriu 10 enters en un fitxer de text i després els llegeix, mostrant-los de nou per pantalla.



BlueJ: exemplesT4

```
import java.io. File;
import java.io. FileNotFoundException;
import java.io. PrintWriter;
import java.util. Scanner;

public class TestPrintWriterScanner {
    private TestPrintWriterScanner() { }

    public static void main(String[] args) {
        String fitxer = "file1.txt";
        try {
            PrintWriter pw = new PrintWriter(new File(fitxer));
            for (int i = 0; i < 10; i++) { pw.println(i); }
            pw.close();
            Scanner scanner = new Scanner(new File(fitxer));
            while (scanner.hasNextInt()) {
                System.out.println("Valor llegit: " + scanner.nextInt());
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            System.err.println("Problemes en obrir el fitxer " + fitxer);
        }
    }
}
```

Tancament de fitxers

- Els fitxers sempre han de ser explícitament tancats després de ser utilitzats. En particular, un fitxer sobre el que s'ha escrit, ha de ser tancat abans de poder ser obert per a lectura (per garantir l'escriptura de dades en el disc).
- Si el programador no tanca de forma explícita el fitxer, Java ho farà automàticament en acabar l'execució del programa, però si el programa acaba anormalment (se'n va la llum!), el fitxer pot estar incomplet o corrupte.
- Les classes `Scanner` i `PrintWriter` tenen el mètode:

```
public static void close()
```
- Quan s'obrin i manipulen fitxers en una instrucció `try-catch` i no es pot garantir que tots els fitxers queden tancats en qualsevol circumstància, s'usa un bloc `finally` per tancar-los.

Tancament de fitxers - Exemple

- Programa que crea una còpia de seguretat d'un fitxer: crea una còpia i la deixa de només lectura; si la còpia no es pot crear, el programa acaba sense fer res més.

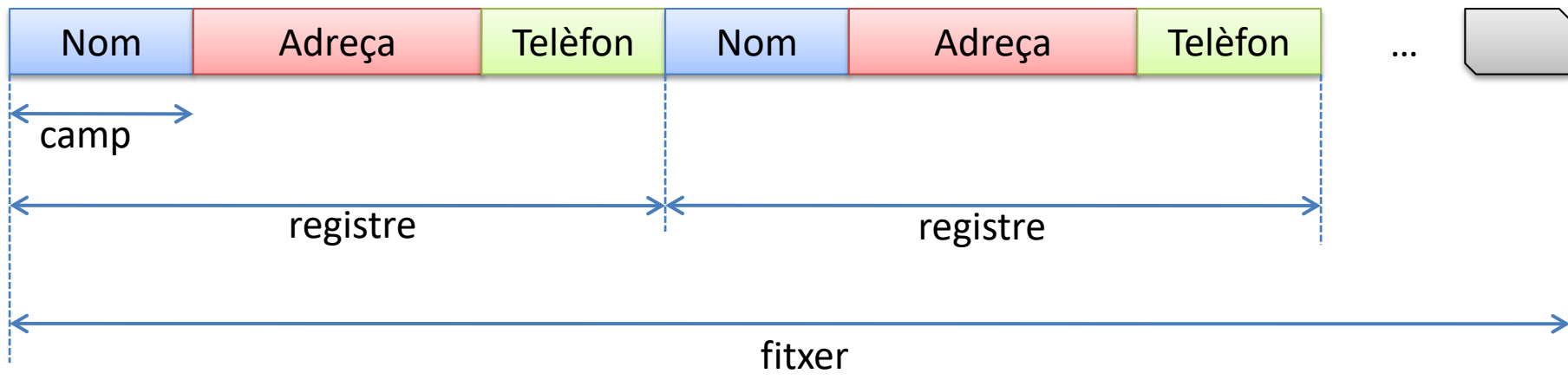
```
import java.io. File;
import java.io. FileNotFoundException;
import java.io. PrintWriter;
import java.util. Scanner;
public class CopiaSeguretat {
    private CopiaSeguretat() { }
    public static void main(String[] args) {
        File fE = new File("file1.txt"), fS = new File("file1Back.txt");
        File f = null; Scanner sc = null; PrintWriter pw = null;
        try {
            f = fE; sc = new Scanner(f);
            f = fS; pw = new PrintWriter(f);
            copia(sc, pw);
            // sc.close(); pw.close();
            // fS.setReadOnly();
        } catch (FileNotFoundException e) {
            System.err.println("Error " + e + " en obrir " + f);
        } finally {
            if (sc != null) { sc.close(); }
            if (pw != null) {
                pw.close();
                fS.setReadOnly(); // permís de només lectura
            }
        }
    }
    ...
}
```



BlueJ: exemplesT4

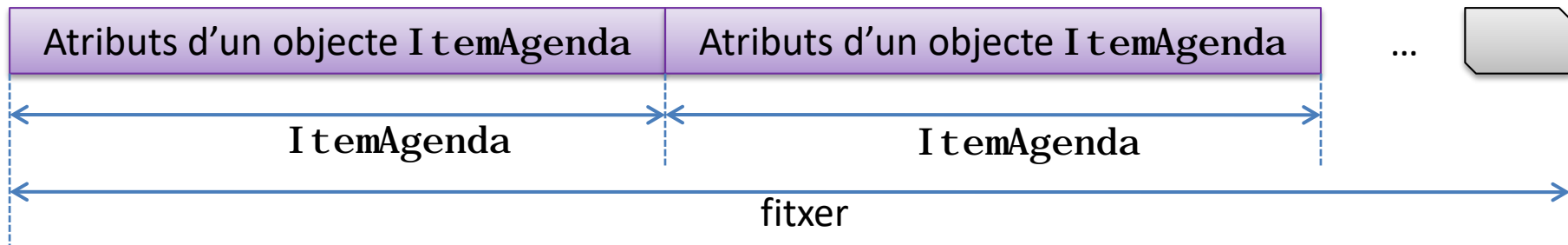
Fitxers binaris

- En lloc d'emmagatzemar dades en fitxers de text (escrivint la seua representació textual), també es possible emmagatzemar i recuperar, de manera codificada, dades en fitxers binaris (així, un `int` ocuparà 4 bytes, un `double` 8, etc.).
- En general s'ha de seguir una política en l'emmagatzemament que facilite després la recuperació adequada de la informació.
- Per exemple: si es salva el contingut d'una agenda, emmagatzemant els valors elementals de cadascun dels seus components, es pot seguir una política d'agrupació de la informació en els anomenats **registres** (grups d'informació semblant que es repeteixen):

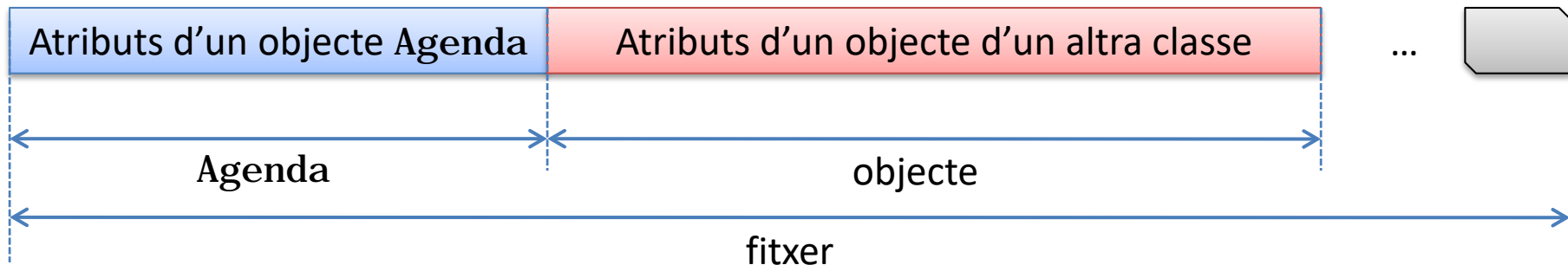


Fitxers binaris d'accés seqüencial

- En Java també és possible emmagatzemar una seqüència d'objectes per poder recuperar-los posteriorment en el mateix ordre. Això s'anomena **serialització** i es parla de **fitxers binaris d'accés seqüencial (fbs)**.
- Per exemple, és possible salvar el contingut d'una agenda (objecte de la classe Agenda) emmagatzemant un a un els seus objectes constituents (objectes de tipus ItemAgenda).



- O fins i tot es podria salvar en el fitxer un objecte de tipus Agenda (que inclouria internament tots els ItemAgenda que el formen).
- A més, un fitxer pot contenir objectes pertanyents a diferents classes o fins i tot incloure objectes i valors elementals simultàniament.



Fitxers binaris d'accés seqüencial

- En Java és possible escriure o llegir-hi en format binari :
 - Valors de tipus primitius com `boolean`, `int`, `double`, etc.,
 - Objectes, amb tots els seus components interns, incloent tots els objectes referenciats pels primers (i així successivament, de manera recursiva).
- S'utilitzen per a això les classes `java.io.ObjectInputStream` (lectura) i `java.io.ObjectOutputStream` (escriptura) del paquet `java.io`.

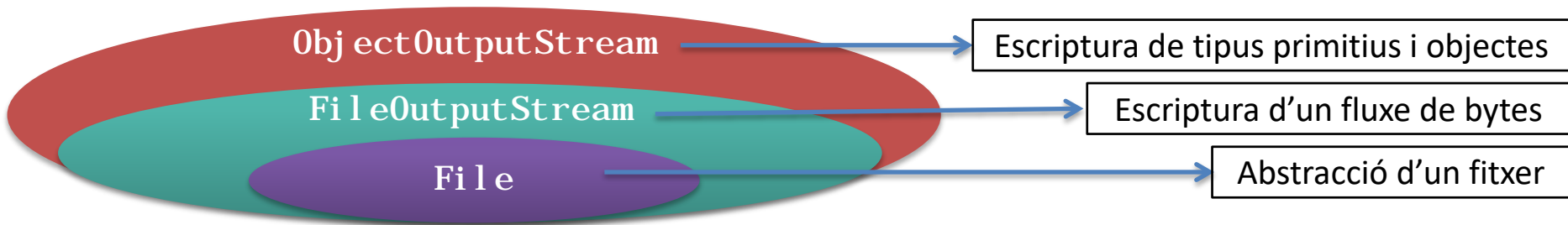


Atenció: per poder escriure o llegir els objectes d'una classe en un stream és necessari que en la declaració d'aquesta classe s'assenya-le que **implementa** l'interfície **Serializable**, el que s'aconsegueix declarant-ho en la seua capçalera. Per exemple :

```
public class ItemAgenda implements Serializable {  
    .....  
} // fi de la classe ItemAgenda
```

Fitxers binaris d'accés seqüencial

- Per a la **lectura** o **escriptura** d'un fbs s'haurà de:
 1. Crear un `File` amb l'**origen/destí** de les dades.
 2. Embolicar-lo en un `FileInputStream/FileOutputStream` per a crear un fluxe de dades **des de/cap al** fitxer.
 3. Embolicar l'objecte anterior en un `ObjectInputStream/ObjectOutputStream` per a poder **llegir/escriure** tipus de dades primitius u objectes del fluxe de dades.
- Posteriorment, per escriure o llegir s'han d'usar mètodes com:
 - `writeInt`, `writeDouble`, `writeBoolean`, `writeObject`, etc. (en **escriptura**)
 - `readInt`, `readDouble`, `readBoolean`, `readObject`, etc. (en **lectura**)



- Exemple d'instanciació per escriptura de dades primitives o objectes a partir d'un fitxer binari (la lectura és anàloga):

```
ObjectOutputStream out = new ObjectOutputStream(  
    new FileOutputStream(new File(fitxer)));  
  
out.writeInt(45);
```

És possible evitar la creació del `File`, passant només el nom del fitxer al constructor de `FileOutputStream`



Escriptura i lectura en fbs

- La classe `ObjectOutputStream` conté mètodes per a escriure tipus primitius i objectes a un stream.
- Existeixen mètodes semblants en `ObjectInputStream` per a llegir-los.

Mètode / Constructor	Descripció
<code>ObjectOutputStream(OutputStream out)</code>	Crea un nou objecte a partir del stream d'eixida
<code>void writeInt(int v)</code>	Escriu l'enter <code>v</code> al stream d'eixida com 4 bytes
<code>void writeLong(long v)</code>	Escriu el <code>long v</code> al stream d'eixida com 8 bytes
<code>void writeUTF(String str)</code>	Escriu el <code>String str</code> amb format portable (UTF8 mod.)
<code>void writeDouble(double v)</code>	Escriu el <code>double v</code> al stream d'eixida com 8 bytes
<code>void writeObject(Object obj)</code>	Escriu l' <code>Object obj</code> a l'stream d'eixida. El que provoca l'escriptura de tots els objectes dels quals <code>obj</code> estiga compost (i així recursivament).
<code>void close()</code>	Tanca el <code>ObjectOutputStream</code>

- Tots els mètodes anteriors poden llançar l'excepció **`IOException`**.



Els fitxers binaris creats amb `ObjectOutputStream` només poden ser llegits amb un `ObjectInputStream`, a causa de les conversions de format que s'efectuen.

Gestió d'excepcions en `ObjectInputStream` i `ObjectOutputStream`

- El constructor de `FileOutputStream/FileInputStream` pot llançar l'excepció `FileNotFoundException`.
 - Si el fitxer especificat no existeix en el sistema de fitxers.
- Els mètodes d'escriptura de `ObjectOutputStream` i els mètodes de lectura de `ObjectInputStream` (inclòs el mètode `close()` d'ambdues classes) poden llançar alguna excepció de tipus `IOException` o d'alguna subclasse de la mateixa.
 - Si es donen problemes en escriure el fitxer (permisos, hardware, etc.).
 - Si la classe de l'objecte que es vol escriure no és serialitzable.
- A més, el mètode `readObject()` de la classe `ObjectInputStream` pot llançar una `ClassNotFoundException` si no es pot determinar la classe de l'objecte que s'està llegint.
- Totes aquestes excepcions han de ser gestionades mitjançant l'ús adequat d'un bloc `try-catch`.

Exemple d'ús de fbs de valors elementals

- Programa que escriu dades en un fbs i després les llegeix i les mostra per pantalla.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
public class Qualificacions {
    private Qualificacions() { }
    public static void main(String[] args) {
        String fitxer = "qualificacions.dat", nom = "PRG";
        int conv = 1; double nota = 7.8;
        try {
            ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(fitxer));
            out.writeUTF(nom); out.writeInt(conv); out.writeDouble(nota);
            out.close();
            ObjectInputStream in = new ObjectInputStream(new FileInputStream(fitxer));
            System.out.println("Valor llegit de nom: " + in.readUTF());
            System.out.println("Valor llegit de convocatòria: " + in.readInt());
            System.out.println("Valor llegit de nota: " + in.readDouble());
            in.close();
        } catch (FileNotFoundException e) {
            System.err.println("Problemes amb el fitxer " + fitxer + "." + e.getMessage());
        } catch (IOException e) {
            System.err.println("Problemes en escriure/llegir al/del fitxer " + fitxer);
        }
    }
}
```


Exemple d'ús de fbs d'objectes

- Com a exemple, s'utilitzarà l'escriptura i lectura d'objectes en un fitxer per emmagatzemar i recuperar els valors d'una agenda.
- Per a l'exemple, s'utilitzaran les classes:
 - `ItemAgenda` (informació individual d'un contacte a l'Agenda),
 - `Agenda` (informació de tots els contactes).

```
import java.io.Serializable;
public class ItemAgenda implements Serializable {
    private String nom; private String tel; private int postal;

    public ItemAgenda(String n, String t, int p) {
        nom = n; tel = t; postal = p;
    }

    public String toString() {
        return nom + ": " + tel + " (" + postal + ")";
    }

    // altres mètodes ...

} // fi de la classe ItemAgenda
```

Exemple d'ús de fbs d'objectes

- Cal recordar que hi ha que dir que els objectes que es van a guardar són “**serialitzables**”, el que es fa declarant-ho a les capçaleres de les classes.

```
import java.io.*;
public class Agenda implements Serializable {
    public static final int MAX = 8;
    private ItemAgenda[] lArray;  private int num;

    public Agenda() { lArray = new ItemAgenda[MAX];  num = 0; }

    public void insertar(ItemAgenda b) {
        if (num >= lArray.length) { dupEspai(); }
        lArray[num++] = b;
    }

    public String toString() {
        String res = "";
        for (int i = 0; i < num; i++) { res += lArray[i] + "\n"; }
        res += "===== ";
        return res;
    }

    // Altres mètodes de la classe ... com esborrar, recuperarPerNom, etc.
```

Exemple d'ús de fbs d'objectes

- A la classe Agenda es defineixen, a més, mètodes per escriure i llegir un objecte Agenda:

```
// en la classe Agenda ...
public void guardarAgenda(String fitxer) {
    try {
        FileOutputStream fos = new FileOutputStream(fitxer);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(this); oos.close();
    } catch (IOException ex) {
        System.err.println("Error en guardar: " + ex.getMessage());
    }
}

public static Agenda recuperarAgenda(String fitxer) {
    Agenda aux = null;
    try {
        FileInputStream fis = new FileInputStream(fitxer);
        ObjectInputStream ois = new ObjectInputStream(fis);
        aux = (Agenda) ois.readObject(); ois.close();
    } catch (IOException ex) {
        System.err.println("Error al recuperar: " + ex.getMessage());
    } catch (ClassNotFoundException ex) {
        System.err.println("Error de tipus: " + ex.getMessage());
    }
    return aux;
}
} // fi de la classe Agenda
```

Exemple d'ús de fbs d'objectes

- En la classe **GestorProva** es crea, guarda i recupera una **Agenda** amb alguns elements :

```
import java.io.*;
public class GestorProva {
    private GestorProva() { }

    public static void main(String[] args) {
        ItemAgenda i1 = new ItemAgenda("Carles Pla", "622115611", 46022);
        ItemAgenda i2 = new ItemAgenda("Olivia", "963221153", 46010);
        ItemAgenda i3 = new ItemAgenda("Joan Duato", "913651228", 18011);

        Agenda a1 = new Agenda();
        a1.insertar(i1); a1.insertar(i2); a1.insertar(i3);

        // Guardar i mostrar l'Agenda a1:
        a1.guardarAgenda("agenda1.dat");
        System.out.println("AGENDA EMMAGATZEMADA: ");
        System.out.println(a1);

        // Recuperar del fitxer i mostrar l'Agenda llegida:
        Agenda rec = Agenda.recuperarAgenda("agenda1.dat");
        System.out.println("AGENDA RECUPERADA: ");
        System.out.println(rec);
    }
} // fi de la classe GestorProva
```

Exemple d'ús de fbs d'objectes

- Quan s'executa el main de GestorProva s'obté el següent :

AGENDA EMMAGATZEMADA:

Carles Pla: 622115611 (46022)

Olivia: 963221153 (46010)

Joan Duato: 913651228 (18011)

=====

AGENDA RECUPERADA:

Carles Pla: 622115611 (46022)

Olivia: 963221153 (46010)

Joan Duato: 913651228 (18011)

=====

- I en el directori de treball es té el fitxer:

```
$ ls -l *.dat
```

```
-rw-r--r-- 1 professor PRG 276 2012-03-20 18:00 agenda1.dat
```

Determinació de la fi del fitxer

- De vegades no es coneix inicialment el nombre d'elements que conté un fitxer.
- Si s'està llegint més enllà de **la fi** d'aquest es produirà una excepció (**IOException**) que permetrà, si és tractada, gestionar la situació.
- A més, en el cas dels **ObjectInputStream**, quan es llegeixen valors es produeix, en intentar accedir més enllà del final del fitxer, l'excepció **EOFException** (subclasse de **IOException**).
- **En conclusió:** és possible tractar tots els elements d'un fitxer llegint-los un a un fins que es produeix l'excepció corresponent.
- L'exemple següent mostra aquesta tècnica que, a més, es pot utilitzar en qualsevol fluxe que pugui produir el llançament d'una excepció semblant.

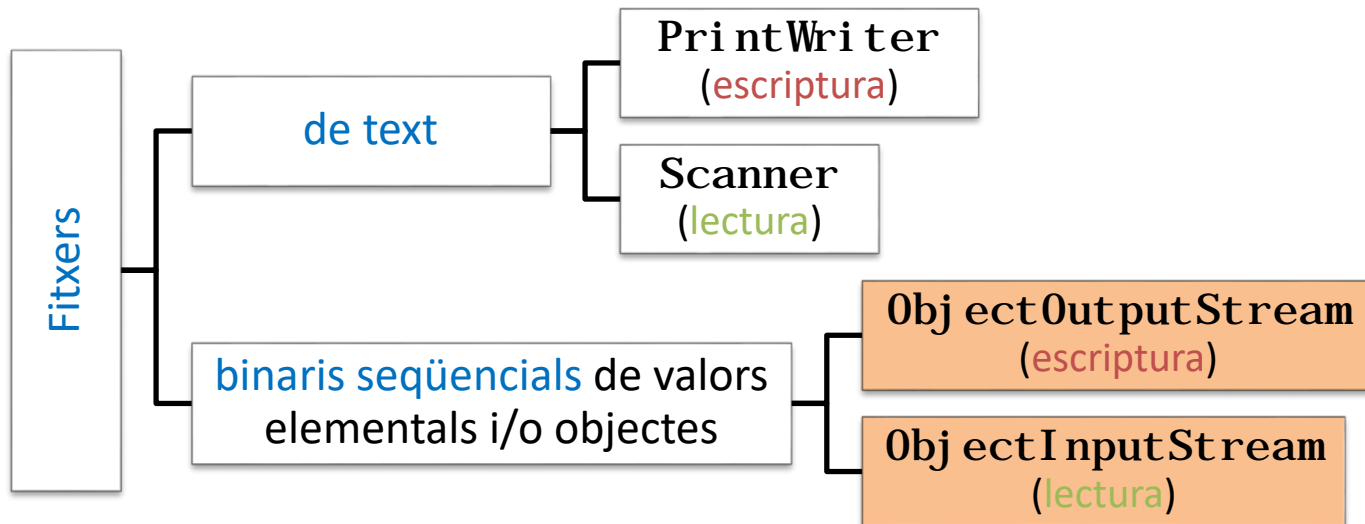
Determinació de la fi del fitxer

- L'exemple següent mostra la lectura d'un fitxer d'int per al qual no és conegut el nombre de valors que conté.
- Els valors llegits s'escriuen un darrere de l'altre a la pantalla i en arribar al final del fitxer s'escriu un avís.

```
public static void llegirFitxerBinariInt(String fitx) {  
    ObjectInputStream ois = null;  
    try {  
        ois = new ObjectInputStream(new FileInputStream(fitx));  
        try {  
            while (true) {  
                int val = ois.readInt();  
                System.out.println(val);  
            }  
        } catch(EOFException ef) {  
            System.out.println("S'ha arribat a la fi del fitxer");  
        }  
    } catch(IOException fex) {  
        System.err.println("Error en recuperar: " + fex.getMessage());  
    } finally {  
        try {  
            if (ois != null) { ois.close(); }  
        } catch(IOException ex) {  
            System.err.println("Error en tancar el fitxer: " + ex.getMessage());  
        }  
    }  
}
```

Resum

- L'ús de fitxers permet que la informació sobrevisca a l'execució dels programes mitjançant l'ús d'emmagatzematge secundari.
- Java permet la creació de fitxers binaris independents de la plataforma i de text usant mecanismes basats en fluxes (*streams*).

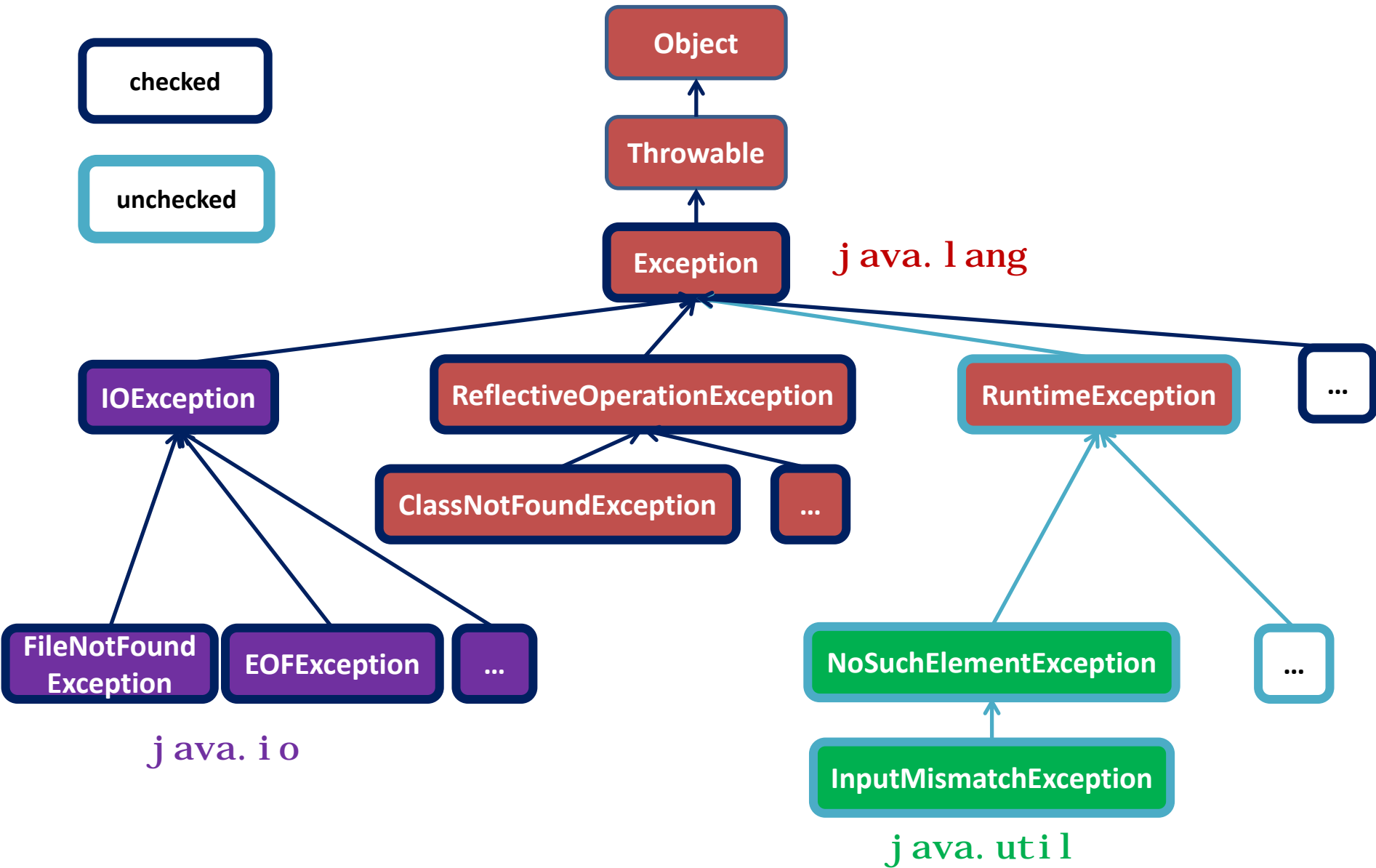


Recomanacions al treballar amb fitxers

- És important gestionar les excepcions en els processos d'entrada/eixida:
 - Errors generals d'entrada/eixida (**IOException**) i els seus casos particulars:
 - Fitxers inexistents o amb problemes (**FileNotFoundException**).
 - Condició de fi de fitxer (**EOFException**).

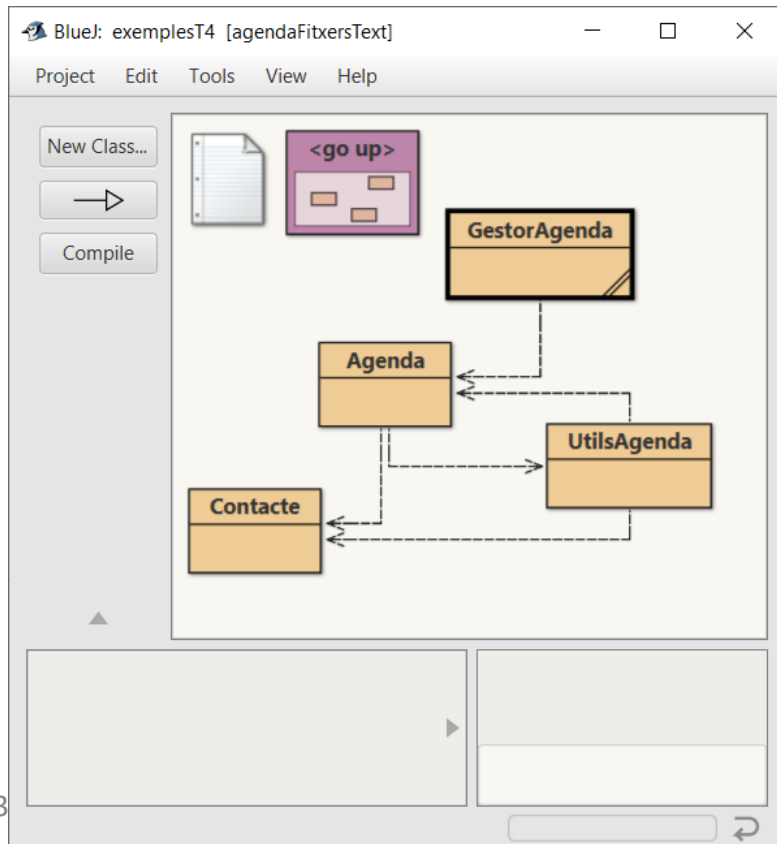
Son classes d'excepcions *checked*, el compilador impedeix que el programador s'oblidi d'elles o les ignore: perquè el programa compile correctament, ha de capturar-les o propagar-les explícitament.
 - Incoherències de tipus en usar:
 - Scanner (**InputMismatchException**).
 - El mètode `readObject()` de la classe `ObjectInputStream` (**ClassNotFoundException**).
- Els fluxes sempre han de tancar-se explícitament després de ser utilitzats. Totes les classes presentades tenen un mètode `close()` per tancar el fluxe.

Jerarquia d' excepcions en entrada/eixida



Example: paquet agendaFitxersText

- Revisa les classes del paquet agendaFitxersText del projecte **BlueJ** **exemplesT4**. En particular, els mètodes carregarFitxerText(Scanner) i guardarFitxerText(PrintWriter) de la classe Agenda i, en la instrucció switch del main de la classe GestorAgenda, els case 4 i 5.
- Després, executa el main de la classe GestorAgenda, triant l'opció 4 del menú per carregar el fitxer de text "agendaFitxersText/agenda.txt".
- Prova les diferents opcions del menú de l'aplicació.



The screenshot shows a text file named 'agenda.txt' with the following content:

```
PEPE MIRO: 123456789
SARA NAVARRO: 987654321
MARIA PALANQUES: 456324568
JOAN MIRALLES: 987654345
PEPE SANCHEZ: 123456544
ANNA FOLCH: 456321890
```

The screenshot shows the BlueJ Terminal Window for the project 'exemplesT4'. It displays a menu with the following options:

```
MENÚ
-----
| 1.- Mostrar contactes          |
| 2.- Afegir un contacte        |
| 3.- Consultar un contacte     |
| 4.- Carregar agenda des de fitxer |
| 5.- Guardar agenda en fitxer   |
| 0.- Finalitzar                |
-----
Tria una opcio:
```

At the bottom, there is a text input field with the prompt 'Type input and press Enter to send to pro'.

Exercici: classe EscriuNums

- Completa el codi del main de la classe EscriuNums al paquet fitxersText del projecte *BlueJ exercicisT4*. Executa'l i comprova el contingut del fitxer "fitxersText/test1.txt".



BlueJ: exercicisT4 [fitxersText]

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;

/**
 * Escriu en un fitxer de text (test1.txt) els numeros de 1'1 al 999.
 * @author PRG
 * @version Curs 2019/20
 */
public class EscriuNums {
    private EscriuNums() { }

    public static void main(String[] args) {
        String fitxer = "fitxersText/test1.txt";

        try {
            System.out.println("Escrivint fitxer " + fitxer);
            PrintWriter pw = /* COMPLETAR - crear el PrintWriter de fitxer */;
            for (int i = 1; i < 1000; i++) {
                /* COMPLETAR - Escriure en pw */
            }
            /* COMPLETAR - Tancar pw*/
        } catch (FileNotFoundException ex) {
            System.err.println("Problemes en obrir el fitxer "
                + fitxer + ". Rao: " + ex.getMessage());
        }
    }
}
```

Exercici: classe EscriuNumsi Comprova

- Completa el codi del `main` de la classe `EscriuNumsi Comprova` al paquet `fitxersText` del projecte **BlueJ exercicisT4**



BlueJ: exercicisT4 [fitxersText]

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;

/**
 * Escriu en un fitxer de text (test1.txt) els numeros de l'1 al 999.
 * Posteriorment, els torna a llegir d'aquest fitxer per tal de sumar-los,
 * verificant que el resultat es correcte.
 * @author PRG
 * @version Curs 2019/20
 */
public class EscriuNumsiComprova {
    private EscriuNumsiComprova() { }

    public static void main(String[] args) {
        String fitxer = "fitxersText/test1.txt";
        int suma = 0;
        try {
            System.out.println("Escrivint fitxer " + fitxer);
            PrintWriter pw = /* COMPLETAR - crear el PrintWriter de fitxer */;
            for (int i = 1; i < 1000; i++) {
                /* COMPLETAR - Escriure en pw */
            }
            /* COMPLETAR - Tancar pw*/

            System.out.println("Llegint del fitxer " + fitxer);
            Scanner scanner = /* COMPLETAR - crear l'Scanner de fitxer */
```

Exercici: classe Filtra

- Completa el codi de la classe Filtra al paquet fitxersText del projecte **Bluej exercicisT4** i prova a executar-lo amb el fitxer "fitxersText/alegria.txt" i la paraula "alegria".

The screenshot displays the BlueJ IDE environment with three windows:

- Code Editor (Bluej: exercicisT4 [fitxersText]):** Contains the Java code for the `Filtra` class. The code prompts the user for a file path and a word, then searches for occurrences of that word in the specified file. Comments indicate where to complete the `Scanner` creation and closing.
- Text File (alegria: Bloc de notas):** Shows the content of `alegria.txt`, which consists of several lines of text in Catalan, including "Tinc una alegria sense fronteres...", "D'una esperança que no s'ha acabat...", "No cal plorar, són camins d'amor i llibertat.", and "Una alegria de camins per fer una alegria salvatge i ardent."
- Terminal Window (Bluej: Terminal Window - exercicisT4-Sol):** Displays the output of the program. It shows the search results for the word "alegria" in the file `fitxersText/alegria.txt`, listing line numbers and the corresponding text segments.

```
public class Filtra {  
    private Filtra() { }  
  
    public static void main(String[] args) {  
        if (args.length != 2) {  
            System.out.println("Introdueix la ruta a un fitxer de text "  
                               + "i la paraula a buscar");  
        }  
        else {  
            String filePath = args[0];  
            String paraula = args[1];  
            System.out.println("Buscant les ocurrences de " + paraula  
                               + " en el fitxer: " + filePath);  
            Scanner s = null;  
            try {  
                /* COMPLETAR - crear l'Scanner del fitxer filePath */  
                cercaParaula(s, paraula);  
            } catch (FileNotFoundException ex) {  
                System.out.println("El fitxer " + filePath + " no existeix.");  
            } finally {  
                /* COMPLETAR - tancar s */  
            }  
        }  
    }  
}
```

alegria: Bloc de notas

Archivo Edición Formato Ver Ayuda

ALEGRIA - EL DILUVI

Tinc una alegria sense fronteres,
una alegria plana, sense normes.
Tinc una alegria de camins oberts,
una alegria de color verd.

D'una esperança que no s'ha acabat
d'una esperança...
Queda esperança quan vaig avançant
queda esperança i...

No cal plorar,
són camins d'amor i llibertat.
No cal plorar,
són camins de lluita i llibertat.

Una alegria de camins per fer
una alegria salvatge i ardent.
Una alegria que crema com el foc,
una alegria de color groc.

Bluej: Method Call

void main(String[] args)

Filtra.main({"fitxersText/alegria.txt", "alegria"})

Aceptar Cancelar

Bluej: Terminal Window - exercicisT4-Sol

Options

Buscant les ocurrences de alegria en el fitxer: fitxersText/alegria.txt

3: Tinc una alegria sense fronteres,
4: una alegria plana, sense normes.
5: Tinc una alegria de camins oberts,
6: una alegria de color verd.
18: Una alegria de camins per fer
19: una alegria salvatge i ardent.
20: Una alegria que crema com el foc,
21: una alegria de color groc.

Exercici: classe Elimina

- Completa el codi de la classe Elimina al paquet fitxersText del projecte **BlueJ exercicisT4**. Prova a executar-lo amb el fitxer “fitxersText/prova.txt” i la paraula “el” i comprova el contingut del fitxer resultat “fitxersText/prova_2.txt”



BlueJ: exercicisT4 [fitxersText]

```
public static void elimina(Scanner s, String w, PrintWriter pw) {
    while (/* COMPLETAR */) {
        String p = /* COMPLETAR - llegir una paraula */
        /* COMPLETAR - si p no coincideix amb w, escriure p en pw */
    }
}

public static void main(String[] args) {
    if (args.length != 2) {
        System.out.println("Introdueix la ruta a un fitxer de text "
            + "i la paraula a eliminar");
    }
    else {
        String f1 = args[0];
        String paraula = args[1];
        System.out.println("Eliminant totes les ocurrences de " + paraula
            + " al fitxer: " + f1);
        try {
            Scanner s = /* COMPLETAR - crear l'Scanner del fitxer f1 */;
            File f2 = new File(f1.substring(0, f1.indexOf('.')) + "_2.txt");
            PrintWriter pw = /* COMPLETAR - crear el PrintWriter del fitxer f2 */;
            elimina(s, paraula, pw);
            System.out.println("Resultat en " + f2);
        } catch (FileNotFoundException ex) {
            System.out.println("El fitxer " + f1 + " no existeix.");
        } finally {
            /* COMPLETAR - tancar s */
            /* COMPLETAR - tancar pw */
        }
    }
}
```

prova: Bloc de notas

Archivo Edición Formato Ver Ayuda

el elefant es el mes gran
el gos es el mes menut

Window: Línea 1, c 100%

BlueJ: Method Call

void main(String[] args)

Elimina.main(["fitxersText/prova.txt", "el"])

Aceptar Cancelar

prova_2: Bloc de notas

Archivo Edición Formato Ver Ayuda

elefant es mes gran gos es mes menut

Window: Línea 1, c 100%

Exercici: classe Estadístiques

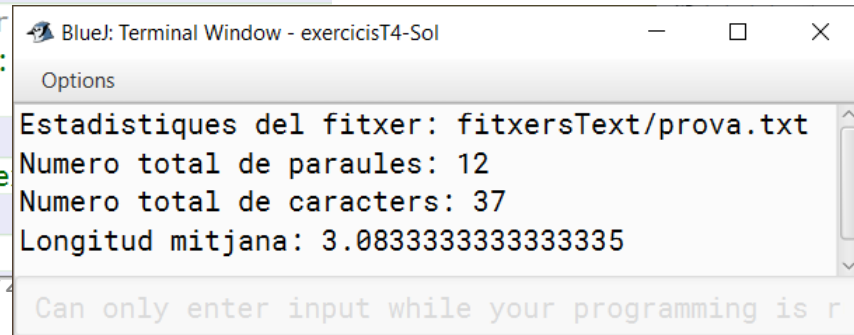
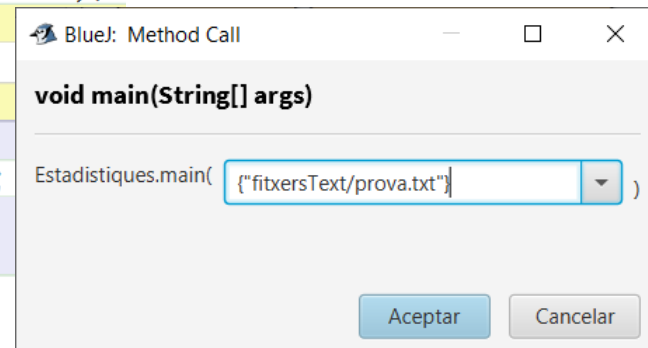
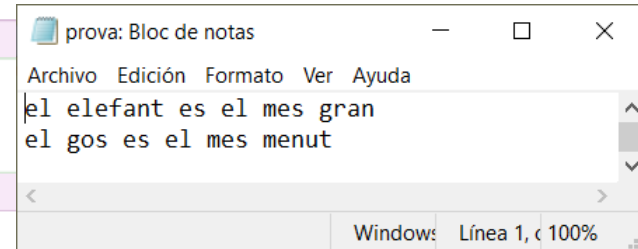
- Completa el codi de la classe Estadístiques al paquet fitxersText del projecte **BlueJ exercicisT4**. Prova a executar-lo amb el fitxer “fitxersText/prova.txt” i comprova què es mostra per pantalla.



BlueJ: exercicisT4 [fitxersText]

```
public static void estadistiques(Scanner s) {  
    int nParaules = 0, nChars = 0;  
    while (/* COMPLETAR */) {  
        String p = /* COMPLETAR - llegir una paraula */  
        nParaules++;  
        nChars += p.length();  
    }  
    System.out.println("Numero total de paraules: " + nParaules);  
    System.out.println("Numero total de caracters:" + nChars);  
    System.out.println("Longitud mitjana: " + nChars / (double) nParaules);  
}
```

```
public static void main(String[] args) {  
    if (args.length != 1) {  
        System.out.println("Introdueix la ruta a un fitxer de text");  
    }  
    else {  
        String f1 = args[0];  
        Scanner s = null;  
        try {  
            s = /* COMPLETAR - crear l'Scanner del fitxer */  
            System.out.println("Estadistiques del fitxer: " + f1);  
            estadistiques(s);  
        } catch (FileNotFoundException ex) {  
            System.out.println("El fitxer " + f1 + " no existeix");  
        } finally {  
            /* COMPLETAR - tancar s */;  
        }  
    }  
}
```



Exercici: classe IOPresio

- Completa el codi dels mètodes escriuPresions(double[][], String) i lligPresions(String, int) de la classe IOPresio al paquet fitxersText del projecte **Bluej exercicisT4**. Després, executa el main i comprova que el contingut dels fitxers "presio.txt" i "presio2.txt" és el mateix.



Bluej: exercicisT4 [fitxersText]

```
/**
 * Escriu en un fitxer de text nomFitx els valors de tipus double
 * emmagatzemats en una matriu presions.
 * Cada fila de la matriu s'escriu en una línia del fitxer i
 * els valors de cada fila s'escriuen separats per espais en blanc.
 * @param presions double[][]
 * @param nomFitx String
 */
public static void escriuPresions(double[][] presions, String nomFitx) {
    PrintWriter pw = null;
    try {
        pw = new /*COMPLETAR*/(new /*COMPLETAR*/(nomFitx));
        for (int i = 0; i < presions.length; i++) {
            for (int j = 0; j < presions[i].length; j++) {
                pw./*COMPLETAR*/(presions[i][j] + " ");
            }
            pw.println();
        }
    } catch (/*COMPLETAR*/ e) {
        System.out.println(ERR + "escriure en el fitxer " + nomFitx);
    } finally {
        if (pw != null) { pw./*COMPLETAR*/; }
    }
}
```

```
public static void main(String[] args) {
    String nomFitx = "fitxersText/presio.txt";
    double[][] matriuPresions = lligPresions(nomFitx, 10);
    String nomFitx2 = "fitxersText/presio2.txt";
    escriuPresions(matriuPresions, nomFitx2);
}
```

```
/**
 * Llig des d'un fitxer de text nomFitx valors de tipus double
 * (amb el "." com separador decimal) que va emmagatzemant en una
 * matriu resultat de dimensions n x n.
 * @param nomFitx String amb el nom del fitxer.
 * @param n int dimensio de la matriu.
 * @return double[][]
 */
public static double[][] lligPresions(String nomFitx, int n) {
    double[][] aux = new double[n][n];
    Scanner scanner = null;
    try {
        scanner = new /*COMPLETAR*/(new /*COMPLETAR*/(nomFitx)).useLocale(Locale.US);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                try {
                    aux[i][j] = scanner./*COMPLETAR*/;
                } catch (/*COMPLETAR*/ e) {
                    System.out.println("\n***ERROR***: Dada no valida");
                }
            }
        }
    } catch (/*COMPLETAR*/ e) {
        System.out.println(ERR + "accedir al fitxer " + nomFitx);
    } finally {
        if (scanner != null) { scanner./*COMPLETAR*/; }
    }
    return aux;
}
```