

# Programació

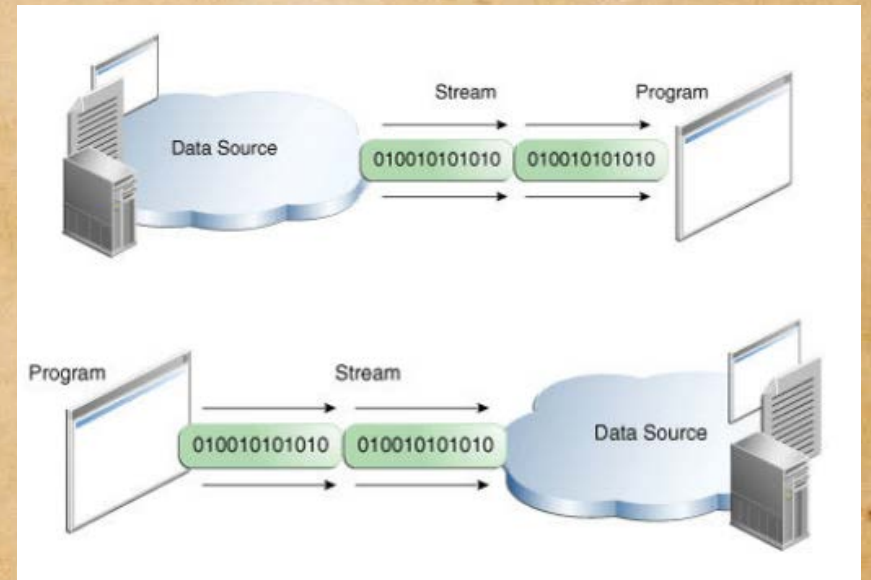


UT12.2. Entrada/Eixida de fitxers de text



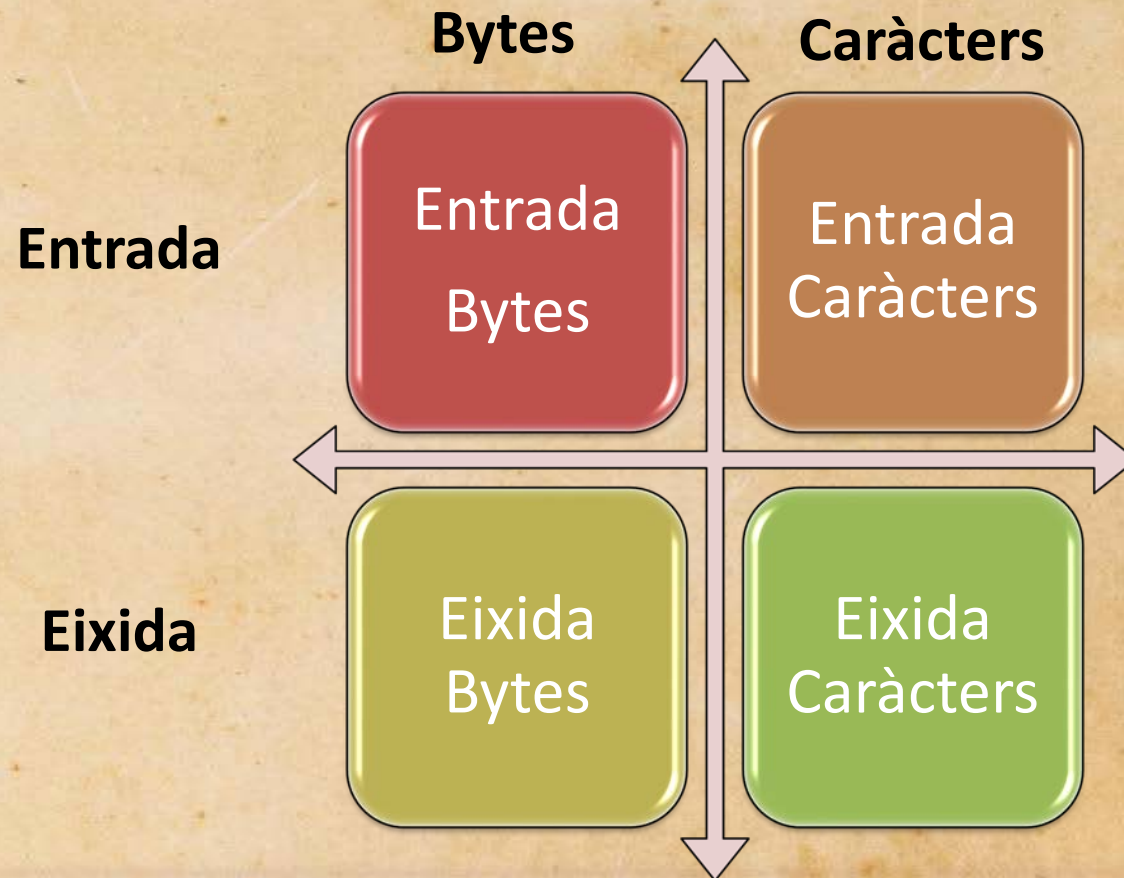
# Introducció: Fluxos de dades.

- Canal de comunicació de les operacions d'entrada/eixida.
- Literalment, és un flux (producció/consum d'informació).
- Ens dona independència:
  - Entrada des del teclat
  - Eixida cap al monitor
  - Lectura d'un fitxer
  - Enviament de dades per xarxa
  - ...





# Introducció: Fluxos de dades.





# Lectura i escriptura de fitxers

- Un objecte de la classe `File` conté la informació de les propietats d'un fitxer/directori o de la seua ruta.
- No obstant això, la classe no conté mètodes per a crear un fitxer o per a escriure o llegir dades a/d'ell.
- Per a realitzar esta entrada/eixida cal crear objectes específics.
- Hi ha dos tipus de fitxers: de text i binaris.



# Tipus de fitxers

- **Fitxers de text:** Les dades es representen com una seqüència de cadenes de text, on cada valor es diferencia usant un delimitador.
- **Fitxers binaris:** Les dades es representen d'acord amb el seu format binari, sense cap separació (els estudiarem més endavant).



# Fitxer binari vs de text

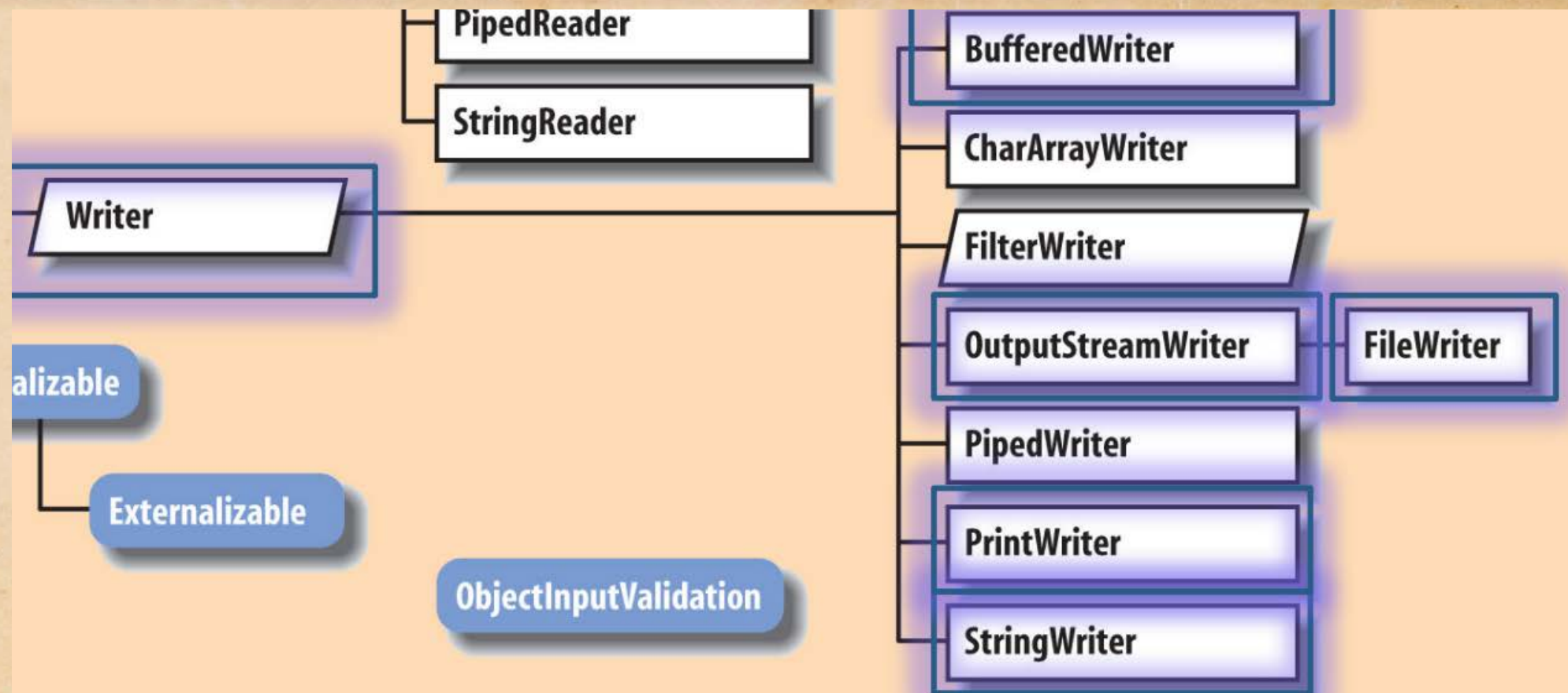
**Fichero binario**

0	00000000	Un número entero: 14
1	00000000	
2	00000000	
3	00001110	
4	00000000	Otro número entero: 33
5	00000000	
6	00000000	
7	00100001	
...	...	

**Fichero de texto**

0	00110001	'1' (código ASCII 0x31)
1	00110100	'4' (código ASCII 0x34)
2	01101000	'h' (código ASCII 0x68)
3	01101111	'o' (código ASCII 0x6F)
4	01101100	'l' (código ASCII 0x6C)
5	01100001	'a' (código ASCII 0x61)
...	...	

# Fluxos d'eixida de caràcters





# Fluxos d'eixida de caràcters

- La classe `PrintWriter` imprimeix representacions formatades d'objectes en una seqüència com una eixida de text
- Es troba a el paquet `java.io`
- Pot ser usat tant per a crear fitxers com per a escriure en ells.

```
PrintWriter output = new PrintWriter(filename);
```

- El objecte `output` tindrà accés als mètodes `print`, `println` o `printf` per a escriure les dades al fitxer.



# API de PrintWriter

## java.io.PrintWriter

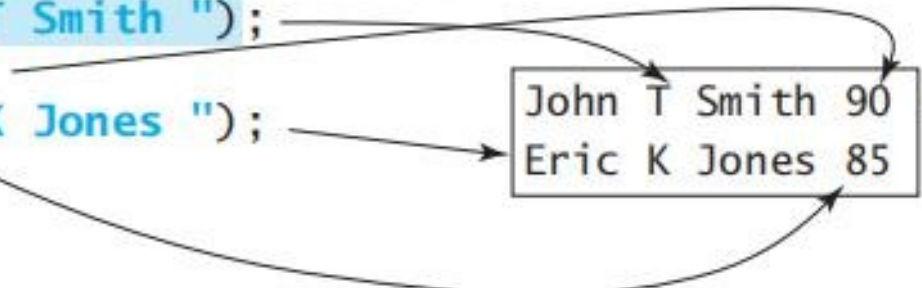
```
+PrintWriter(file: File)
+PrintWriter(filename: String)
+print(s: String): void
+print(c: char): void
+print(cArray: char[]): void
+print(i: int): void
+print(l: long): void
+print(f: float): void
+print(d: double): void
+print(b: boolean): void
```

- Crea un objecte descriptor del fitxer determinat per l'objecte File. Si el fitxer no existeix, el crearà, sinó eliminarà la seva contingut.
- Similar a l'anterior, però es proporciona únicament la ruta al fitxer.
- Escriu en el fitxer la dada donada per paràmetre (hi ha equivalents a println)



# Exemple

```
public class WriteData {  
    public static void main(String[] args) throws IOException {  
        java.io.File file = new java.io.File("scores.txt");  
        if (file.exists()) {  
            System.out.println("File already exists");  
            System.exit(1);  
        }  
        // Create a file  
        java.io.PrintWriter output = new java.io.PrintWriter(file);  
  
        // Write formatted output to the file  
        output.print("John T Smith ");  
        output.println(90);  
        output.print("Eric K Jones ");  
        output.println(85);  
  
        // Close the file  
        output.close();  
    }  
}
```



John T Smith 90
Eric K Jones 85

scores.txt

**Tanca el descriptor !!**  
**Si no es tanca, no es guardaran les dades**



# Estructura try-with-resources

```
try (declare and create resources) {  
    Use the resource to process the file;  
}
```

```
public class WriteDataWithAutoClose {  
    public static void main(String[] args) throws Exception {  
        java.io.File file = new java.io.File("scores.txt");  
        if (file.exists()) {  
            System.out.println("File already exists");  
            System.exit(0);  
        }  
  
        try (  
            // Create a file  
            java.io.PrintWriter output = new java.io.PrintWriter(file);  
        ) {  
            // Write formatted output to the file  
            output.print("John T Smith ");  
            output.println(90);  
            output.print("Eric K Jones ");  
            output.println(85);  
        }  
    }  
}
```

- Per evitar tenir que recordar invocar a **close()**, es fa ús, des de la versió de J D K 7, d'esta estructura.



# Afegir informació a un fitxer existent

- En el cas que necessitem afegir informació sense eliminar el contingut previ d'un fitxer, tindrem que recórrer a la creació d'un objecte de la classe `FileWriter`. Farem servir qualsevol dels dos constructors.

```
FileWriter(File file, boolean append)
```

```
FileWriter(String fileName, boolean append)
```

- El objecte `FileWriter` obtingut, serà el paràmetre que necessitarà ara el constructor de `PrintWriter`. Farem ús d'este constructor:

```
PrintWriter
```

```
public PrintWriter(Writer out)
```



# Exemple - Afegir dades al final

```
try (PrintWriter pw = new PrintWriter(new FileWriter("prueba.txt", true))) {  
    pw.print("Se añade texto al final del fichero");  
} catch (IOException ex) {  
    // Error de entrada / salida  
    ex.printStackTrace();  
}
```



# Altres classes d'eixida de caràcters

- **Writer**: classe abstracta, pare de la majoria dels fluxos de caràcters.
- **FileWriter**: flux que permet escriure en un fitxer, caràcter a caràcter.
- **BufferedWriter**: flux que permet escriure línies de text.
- **StringWriter**: flux que permet escriure en memòria, obtenint allò escrit en un String
- **OutputStreamWriter**: flux que permet transformar un OutputStream en un Writer.
- **PrintWriter**: flux que permet escriure tipus bàsics Java.

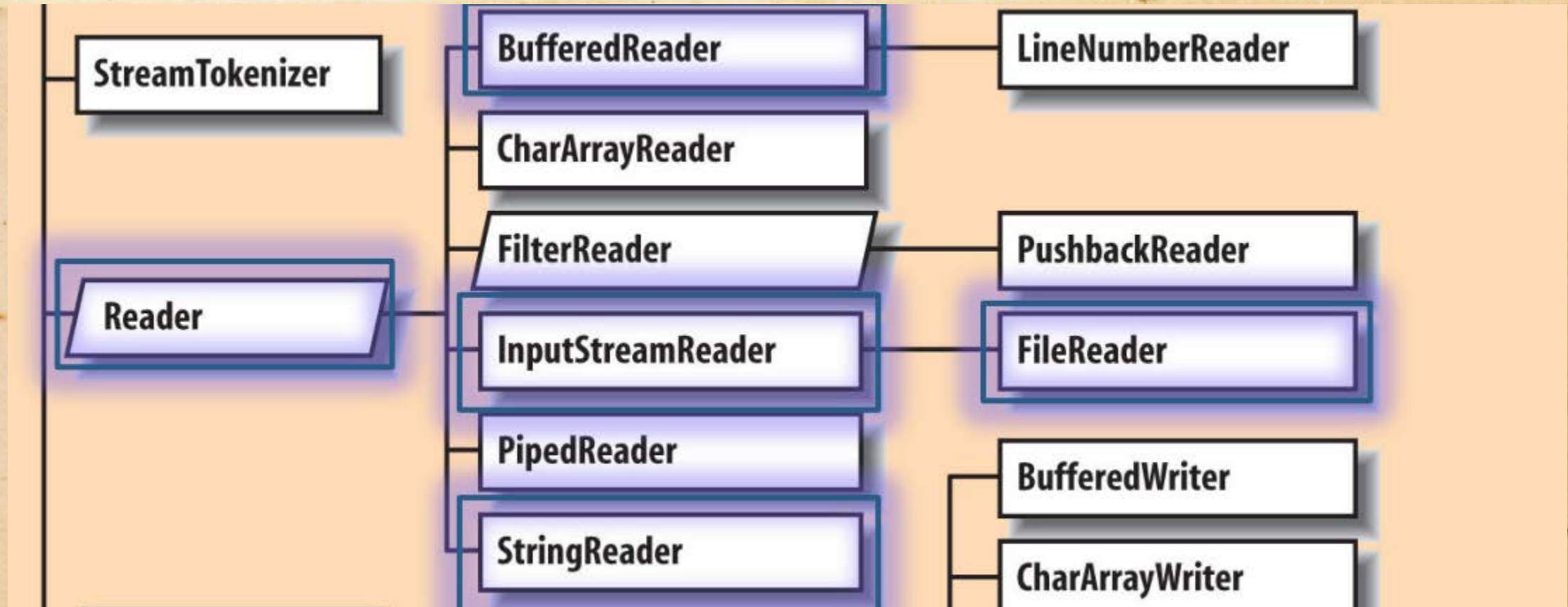
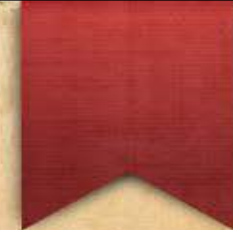


# Exemple – BufferedWriter

```
try (BufferedWriter bw = new BufferedWriter(new FileWriter("fitxer.txt"))) {  
    for (int i = 0; i < 10; i++) {  
        // Escrivim en el buffer  
        bw.write("Escrivint la línia " + (i + 1));  
        // Escrivim un salt de línia  
        bw.newLine();  
    }  
    // Guardem els canvis al fitxer  
    bw.flush();  
  
} catch (IOException e) {  
    System.out.println(e.getMessage());  
}
```



# Fluxos d'entrada de caràcters





# Fluxos d'entrada de caràcters

- Una possibilitat per a realitzar la lectura d'informació es l'ús de la classe `Scanner` que ja coneixem.
- La diferència és que ara, la font de dades no serà `System.in` (entrada estàndard per teclat).
- En comptes d'això, proporcionarem un objecte `File`.

```
Scanner input = new Scanner(new File(filename));
```

**Scanner**

```
public Scanner(File source)  
    throws FileNotFoundException
```

Llança excepció

En comptes de  
**System.in**



# Exemple amb Scanner

```
import java.util.Scanner;

public class ReadData {
    public static void main(String[] args) throws Exception {
        // Create a File instance
        java.io.File file = new java.io.File("scores.txt");

        // Create a Scanner for the file
        Scanner input = new Scanner(file);

        // Read data from a file
        while (input.hasNext()) {
            String firstName = input.next();
            String mi = input.next();
            String lastName = input.next();
            int score = input.nextInt();
            System.out.println(
                firstName + " " + mi + " " + lastName + " " + score);
        }

        // Close the file
        input.close();
    }
}
```

Delimitador per defecte "espai en blanc"

scores.txt

John	T	Smith	90
Eric	K	Jones	85

Tancament del descriptor

També es pot utilitzar try-with-resources



# Scanner - Modificar delimitador

- La classe Scanner també ens permet modificar el delimitador entre tokens llegits

+useDelimiter(pattern: String):  
Scanner

```
try (Scanner fitxer = new Scanner(new File("fitxer.txt"))) {  
    fitxer.useDelimiter(";");  
    while (fitxer.hasNext()) {  
        System.out.println(fitxer.next());  
    }  
} catch (IOException ex) {  
    System.out.println(ex.getMessage());  
}
```



# Pregunta

- Suposant que `test.txt` conté la informació `345 678 9`  
Què s'emmagatzema a les variables `intValue` i `line`?

```
Scanner input = new Scanner(new File("test.txt"));  
int intValue = input.nextInt();  
String line = input.nextLine();
```

- I en este cas, quins valors s'emmagatzemen?

```
Scanner input = new Scanner(System.in);  
int intValue = input.nextInt();  
String line = input.nextLine();
```



# Lectura amb FileReader

- Tal i com passava a “FileWriter”, podem utilitzar els dos constructors de la classe “FileReader”:

```
public FileReader(String fileName)  
    throws FileNotFoundException
```

Creates a new FileReader, given the name of the file to read from.

```
public FileReader(File file)  
    throws FileNotFoundException
```

Creates a new FileReader, given the File to read from.

- “FileWriter” ens permetia escriure caràcter a caràcter sobre un fitxer de text. De manera anàloga “FileReader” ens permet llegir caràcter a caràcter un fitxer de text.



# Exemple amb FileReader

```
try (FileReader fr = new FileReader("fitxer.txt")) {  
    // Cada caràcter es llegit com a un enter (codi ASCII)  
    int ascii = fr.read();  
    // Quan el valor llegit siga un -1 vol dir que ha acabat el fitxer  
    while (ascii != -1) {  
        // Per a obtindre el caràcter associat, hem de fer un casting.  
        System.out.print((char) ascii);  
        ascii = fr.read();  
    }  
} catch (IOException ex) {  
    System.out.println(ex.getMessage());  
}
```



# Lectura amb BufferedReader


- “BufferedWriter” ens permetia escriure línies sobre un fitxer de text. De manera anàloga “BufferedReader” ens permet llegir línies des d’un fitxer de text.

```
try (BufferedReader br = new BufferedReader(new FileReader("fitxer.txt"))) {  
    String linia;  
    while ((linia = br.readLine()) != null) {  
        System.out.println(linia);  
    }  
} catch (IOException ex) {  
    System.out.println(ex.getMessage());  
}
```



# Optimizar la lectura /escritura



- Les classes vistes llegeixen/escriuen físicament en el disc dur. En cas d'escriure o llegir pocs caràcters cada vegada, fa que el procés siga costós i lent, amb molts accessos a disc.
  - Per optimitzar esta lectura/escriptura, hem d'utilitzar les classes Buffered.
  - Internament afegeixen un buffer intermedi que quan es llegeix o escriu controla l'accés a disc.
  - El buffer va emmagatzemant la informació fins que té dades suficients per a accedir a disc.
- 



# Altres classes d'entrada de caràcters

- **Reader**: classe abstracta, pare de la majoria dels fluxos de caràcters.
- **FileReader**: flux que permet llegir d'un fitxer, caràcter a caràcter.
- **BufferedReader**: flux que permet llegir línies de text.
- **StringReader**: flux que permet llegir des de la memòria.
- **InputStreamReader**: flux que permet transformar un `InputStream` en un `Reader`.

```
try {  
    String text = "hola mon";  
    char[] textArray = new char[text.length()];  
    StringReader sr = new StringReader(text);  
    sr.read(textArray);  
    System.out.println(Arrays.toString(textArray));  
} catch (IOException ex) {  
    System.out.println(ex.getMessage());  
}
```