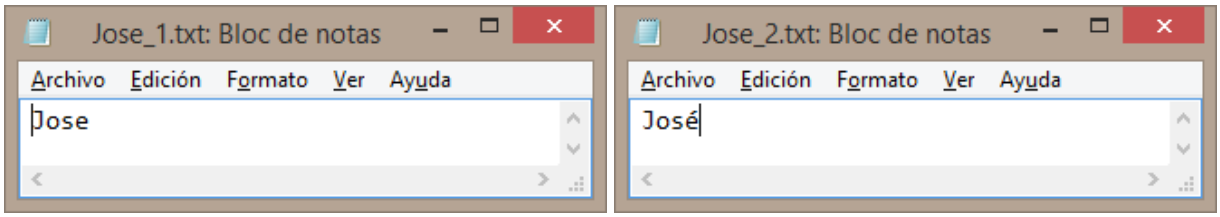
ELS FITXERS DE TEXT Y ELS FITXERS BINARIS

En aquest document anem a aprendre una mica més sobre els fitxers de **text** i **binaris** i com tractar-los des de **java** mitjançant les instruccions de tractament de fitxers que ja vam comentar als documents anteriors.

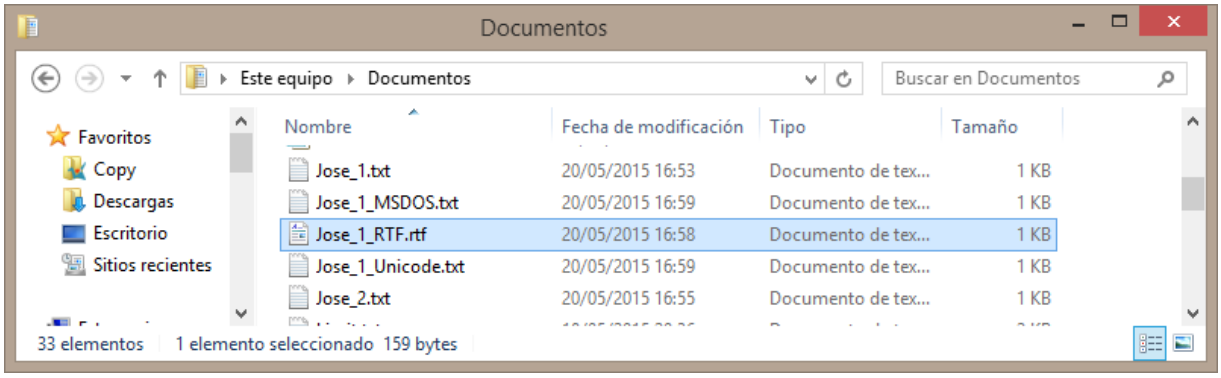
Fitxers de text pla

Son aquells que estan composats únicament per caràcters imprimibles, es a dir, que fent un **type** des de la consola (o **cat** en **Linux**) del fitxer, veurem únicament text que podrem reconèixer (de vegades no el podrem entendre). Alguns editors generen fitxers de document que no son fitxers de text pla, ja que comporten caràcters no imprimibles. Un cas típic de fitxers de text pla son els que tenen extensió **TXT**. El format **RTF** també és un format de text pla, igual que el **HTML** o el **XML**.

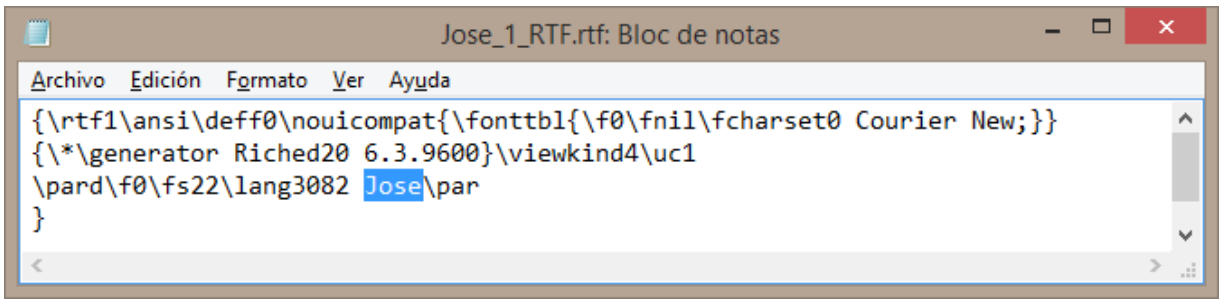
Agafeu la llibreta del Windows (el bloc de notes) i feu dos documents, escrivint en cadascú d'ells únicament **4** caràcters amb el text que s'indica. No poseu cap retorn de carro després de la paraula. Graveu-los al disc, per exemple a **MisDocumentos**.



Aneu al lloc on els hagueu desat i comproveu les propietats del document generat, veureu que en els dos cassos, tenen quatre caràcters. Obriu el primer fitxer amb el **WordPad** i guardeu-lo en format **Document de text**, **Unicode** (És la variant **UTF-16LE**), **Text MSDOS** i **RTF** (amb noms diferents evidentment). Després aneu a comprovar la grandària dels quatre fitxers. Botó dret i propietats. Veureu que la grandària del fitxer depèn del sistema de codificació que heu fet servir.

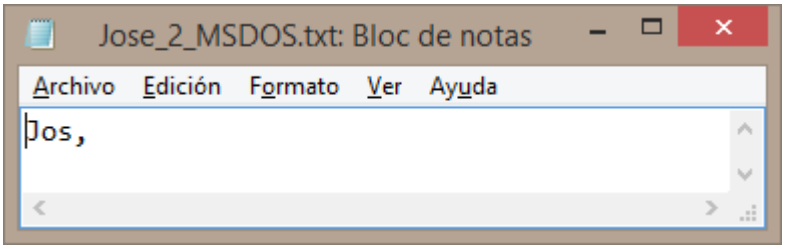


Obrir els quatre fitxers amb el bloc de notes. L'únic que provablement veureu diferent serà el que té l'extensió **rtf**. Entre els altres tres no notareu cap diferència.



Feu el mateix amb el segon fitxer, el que té l'accent a **José**, guardeu-lo en els quatre formats que s'han comentat. Com abans, comproveu la longitud. Una cosa és el que el fitxer ocupa al disc, i una altra la longitud real. Per saber la longitud real, haureu de demanar les propietats per a cadascú dels fitxers creats.

Ara podeu obrir els quatre fitxers amb la llibreta del Windows, i comprovar si hi ha alguna diferència entre ells. El primer que veureu, serà que l'accent de **José**, en el format **MSDOS**, no ha estat correctament interpretat per l'editor.

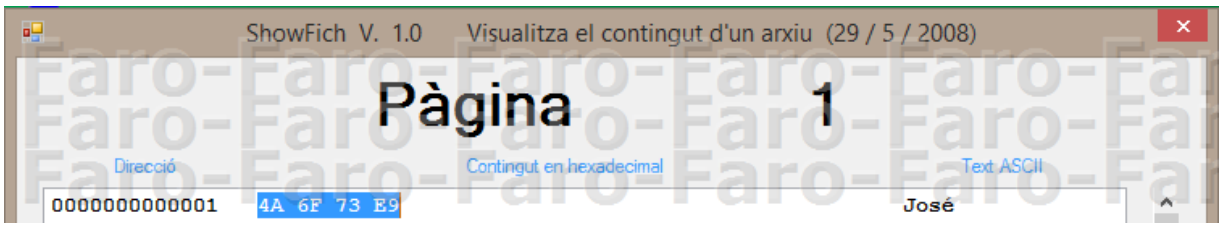


El format RTF, si que es llegeix be, però la seva longitud ha canviat, ara és tres caràcters més llarg, son els **\'** i el **9**.

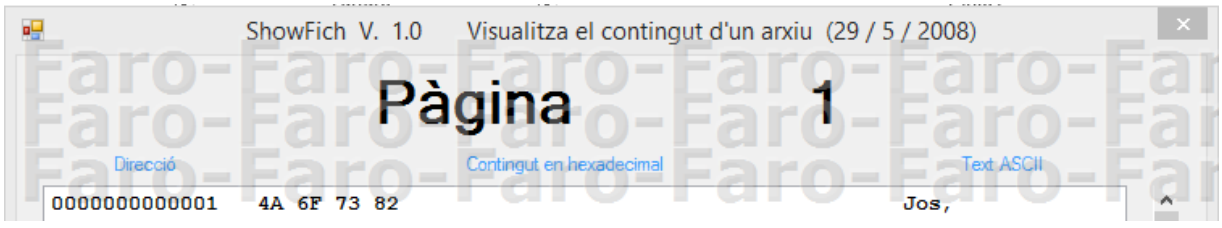
A **Unicode** (sistema **UTF-16)**, per poder representar adequadament tots els caràcters, inclòs en altres idiomes. Es fan servir **dos bytes** per a cada caràcter. El fitxer ocupa dos bytes més, perquè en aquest format, tots els fitxers comencen pels caràcters **FF FE** (notació **hexadecimal**).

Si fem un anàlisi més en profunditat del fitxer, necessitarem un programa que ens permeti analitzar els codis que han agafat els caràcters. Fa anys vaig programar un que faré servir ara.

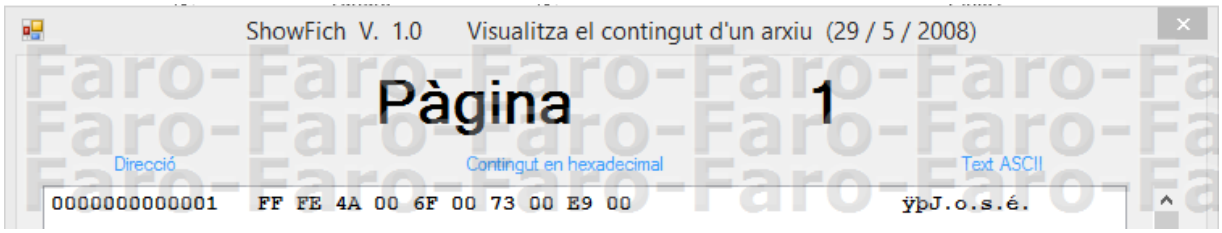
Format de text Windows (ANSI)



Format de text MSDOS (OEM)

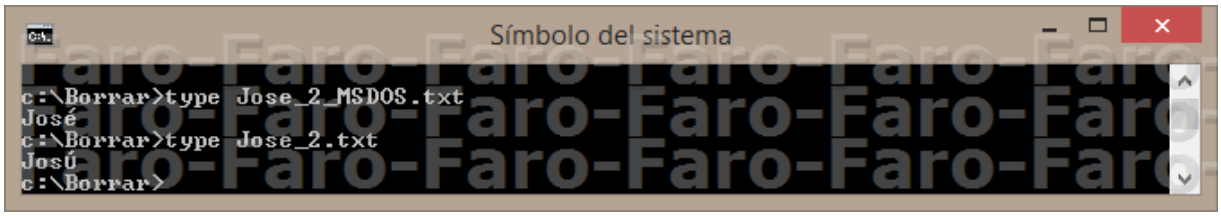


Format de text Unicode (És la variant UTF-16LE (little-endian), amb BOM = FFFE)

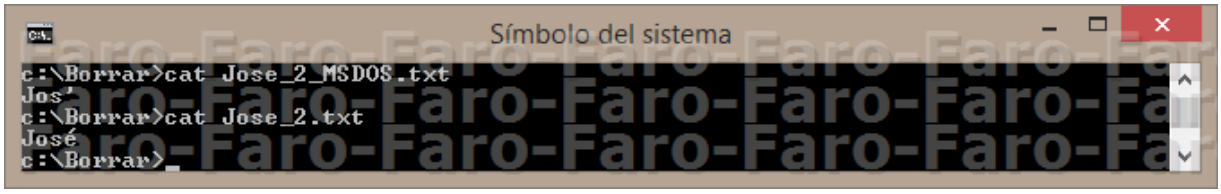


Fixeu-vos que en **Unicode**, el fitxer comença per **FF FE**, apart d'això, cada caràcter s'ha codificat amb el codi **ASCII** corresponent (**Format Windows**), però s'ha afegit un **0** al final. En **MSDOS**, la codificació del caràcter **é** és **82**, i en format Windows és **E9**. En general, els caràcters de l'alfabet anglès (sense accents, **ñ** ni **ç**) es representen amb el mateix codi tant en **MSDOS** com en **Windows** (tècnicament es parla de **OEM** i **ANSI** per diferenciar aquest dos sistemes de codificació).

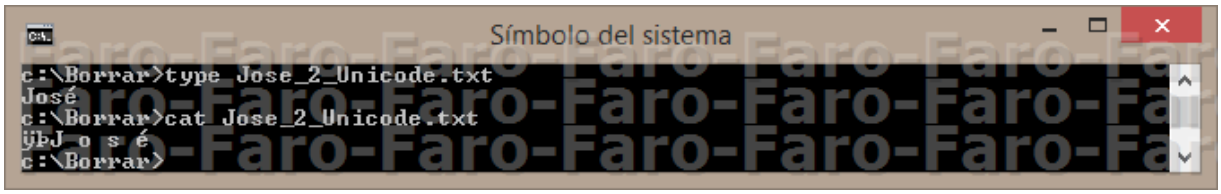
Com a detall interessant, la consola del Windows, fa servir codificació **MSDOS** (**OEM**), per tant, si fem **type** dels dos fitxers, el que es veu ben representat és el que té format **MSDOS**.



Tanmateix, l'ordre **cat** de **Unix** (**Unxutils** instalades sobre **Windows 8.1**), funcionen exàctament a l'inversa, es a dir, reconeixen el format de text **ANSI** i no el **OEM**.



Altre detall interessant és que l'ordre **type** sembla reconèixer correctament el format **Unicode (UTF-16LE)**, mentre que l'ordre **cat** no ho fa (**Unxutils** instalades sobre **Windows 8.1**).



En fi, tot plegat una mica caòtic. Hi ha d'altres sistemes de codificació, en aquesta pàgina podeu trobar més informació: http://czyborra.com/utf/

Gravar text en un fitxer codificant-lo

Hem vist que hi ha molts sistemes de codificació per als fitxers de text. Quin d'aquests es fa servir quan escrivim text en un fitxer des de **java**?. Podem fer servir un sistema de codificació diferent?. Anem a contestar aquestes preguntes a continuació.

```
1  import java.io.*;       // Importem les classes que necessitem
2  public class GravarFormat {
3      // Exemple de gravació de fitxers de text amb codificacions diferents
4      public static void main(String[] args) {
5          try{
6              String s = "äçéÑ€";      // Cadena que escriurem al fitxer
7              // StandardCharsets.
8              PrintWriter pw1 = new PrintWriter("sistema.txt");
9              PrintWriter pw2 = new PrintWriter("utf-8.txt","UTF-8");
10             PrintWriter pw3 = new PrintWriter("utf-16.txt","UTF-16");
11             PrintWriter pw4 = new PrintWriter("ISO_8859_1.txt","ISO_8859_1");
12             PrintWriter pw5 = new PrintWriter("cp1252.txt","cp1252");
13             PrintWriter pw6 = new PrintWriter("ISO646-US.txt","ISO646-US");
14             PrintWriter pw7 = new PrintWriter("ascii.txt","ASCII");
15             PrintWriter pw8 = new PrintWriter("cp850.txt","cp850");
16             pw1.write(s);   pw2.write(s);   pw3.write(s);   pw4.write(s);
17             pw5.write(s);   pw6.write(s);   pw7.write(s);   pw8.write(s);
18             pw1.close();    pw2.close();    pw3.close();    pw4.close();
19             pw5.close();    pw6.close();    pw7.close();    pw8.close();
20         }catch (Exception e){
21             System.out.println("ERROR:   " + e.getMessage());
22         }
23     }
24 }
```

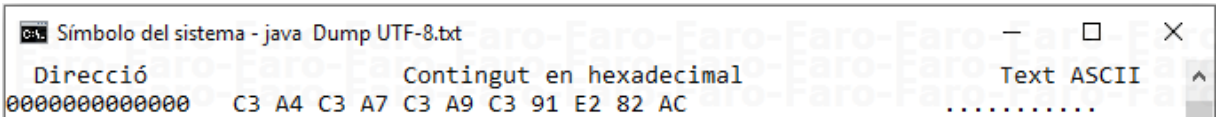
Tot i que hi ha altres formes de fer-ho, el programa anterior fa servir la classe **PrinterWriter** per escriure una cadena en un fitxer amb la codificació que desitgem. S'ha escollit la cadena "**äçéÑ€**". En particular, el símbol del euro, no existeix en alguns dels sistemes de codificació més antics, i per tant no el podrem representar en aquests.

Si no li diem res, **java**, al igual que **python** per defecte, emmagatzema i representa les cadenes en el format **UTF-8**. No obstant, a l'hora de crear els nostres fitxers, tenim diferents opcions per indicar la codificació interna d'aquests entre les vàries existents. Per exemple, les aplicacions gràfiques de Windows (les de consola fan servir altra codificació) fan servir **cp1512**.

Hi ha un número considerablement gran de sistemes de codificació dels caràcters. Molts d'ells fan servir el mateix codi per a les lletres de l'alfabet anglès, però pels caràcters no anglesos, la representació pot canviar d'un sistema de codificació a un altre. Podeu donar un cop d'ull a la pàgina: https://ca.wikipedia.org/wiki/Codificaci%C3%B3_de_car%C3%A0cters

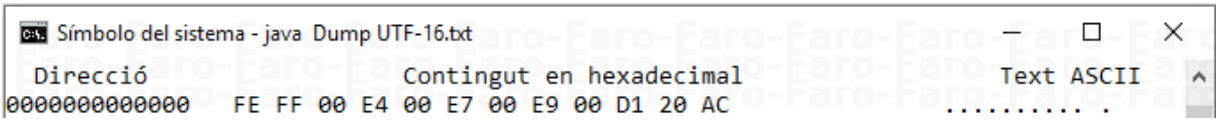
Codificació dels fitxers que hem creat

UTF-8: https://en.wikipedia.org/wiki/UTF-8



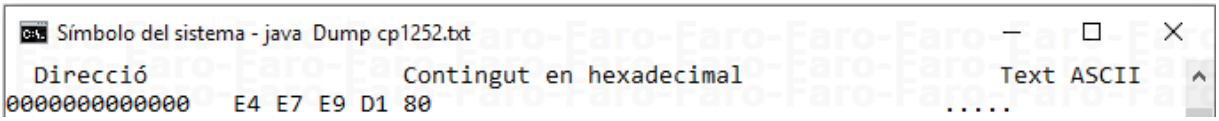
És la codificació per defecte que fa servir **java**, de forma que els fitxers **sistema.txt** i **utf-8.txt** presenten aquesta codificació. **UTF-8** fa servir **de 1 a 4** caràcters per codificar els **1.112.064** símbols que permet. La idea és codificar els símbols més freqüents amb un sol byte, per optimitzar l'espai ocupat pels fitxers i la velocitat d'accés. Té compatibilitat cap enrere amb ASCII, de forma que els 127 símbols americans del **ASCII** de **7 bits** es codifiquen igual (en un byte). Els tres primers caràcters, s'han codificat en **UTF-8** amb **dos bytes** (son caràcters especials), i el símbol del euro amb tres bytes.

UTF-16: https://en.wikipedia.org/wiki/UTF-16

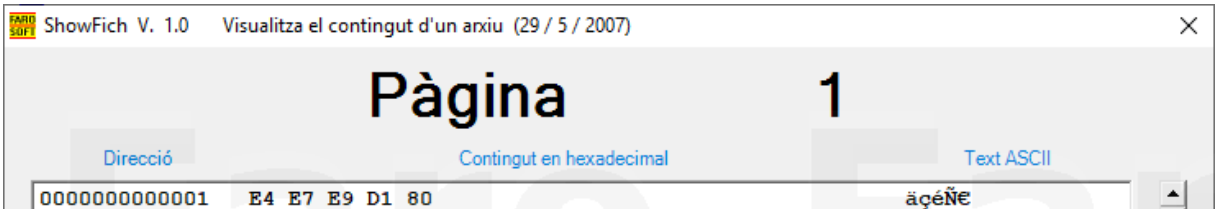


Com s'ha comentat, els fitxers amb aquest sistema de codificació, dediquen els dos primers bytes, que son **FE FF**, a indicar-lo, i després, la majoria dels símbols es codifiquen amb **2 bytes** (fins i tot les lletres de l'alfabet anglès). Té una certa semblança com es veu al sistema **cp1252** que fa servir **Windows**, però invertint dos bytes per a cada caràcter (on el primer byte acostuma a ser zero). El símbol del euro s'ha codificat també amb dos bytes, però en aquesta ocasió, ja no comença amb zero.

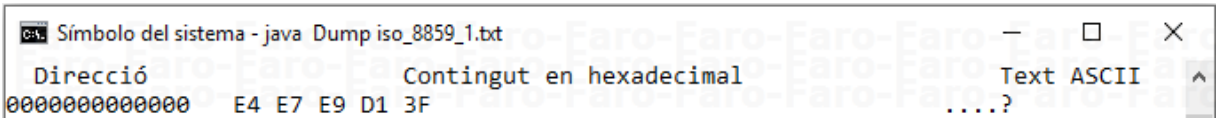
CP1252 o LATIN-1: https://en.wikipedia.org/wiki/Windows-1252



És el sistema de codificació que utilitza Windows en el mode gràfic, de forma que un **Dump** des d'una aplicació **Windows** amb entorn gràfic, representà correctament la cadena de text:

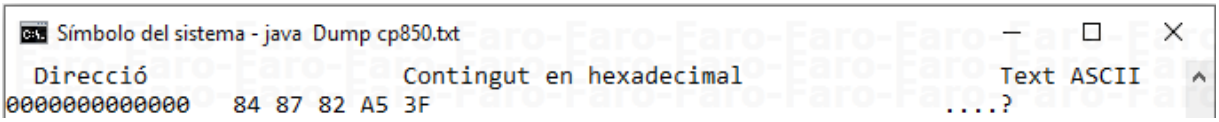


ISO_8859_1: https://en.wikipedia.org/wiki/ISO/IEC_8859-1



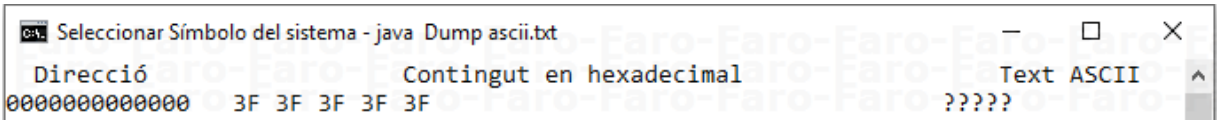
Curiosament, aquest sistema de codificació de caràcters, és molt semblant al **cp1252** del Windows, però no permet representar el símbol del **€** (fixeu-vos que aquesta circumstància s'ha indicat amb el símbol **?**, que és codifica com **3F**).

cp850: https://en.wikipedia.org/wiki/Code_page_850



En aquest sistema de codificació ens passa el mateix. És una mica antiquat, i no s'ha incorporat el símbol del euro als símbols que permet representar. Com al cas anterior, aquest símbol s'ha codificat com **3F = '?'**.

ISO646-US i variant ASCII: https://ca.wikipedia.org/wiki/ISO_646



ASCII es refereix al codi ASCII de 7 bits, que només permet representar 128 caràcters i/o símbols que codifiquen l'alfabet americà (lletres majúscules i minúscules), números i alguns símbols especials (&%$...). En aquests dos jocs de caràcters americans de **7 bits** (**ASCII** e **ISO646-US**), no estan contemplades les lletres accentuades, i el **java** marca amb **3F** (que es tradueix en un **?**), els símbols que no tenen traducció al sistema de codificació de destí. Això ja ho hem comentat amb altres sistemes de codificació per al símbol de **€**, que alguns d'ells no contemplen.

A la pàgina que s'indica, **Microsoft** ens enumera els identificadors dels sistemes de codificació, junt amb els codis de pàgina. Com es pot apreciar, el **Windows-1252** també es reconeix com a **Latin 1**.

https://msdn.microsoft.com/en-us/library/windows/desktop/dd317756%28v=vs.85%29.aspx

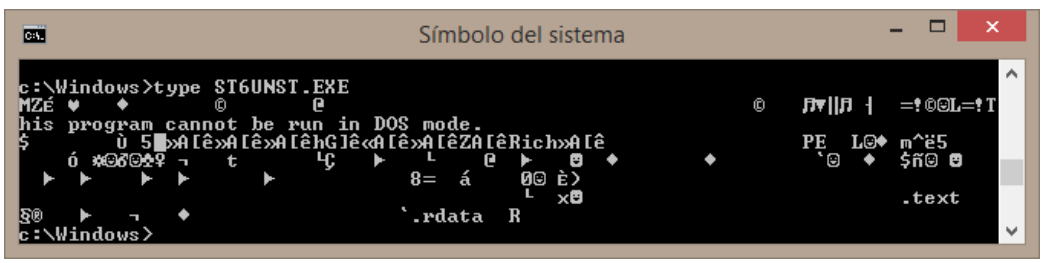
1250	windows-1250	ANSI Central European; Central European (Windows)
1251	windows-1251	ANSI Cyrillic; Cyrillic (Windows)
1252	windows-1252	ANSI Latin 1; Western European (Windows)

En aquesta pàgina, podreu veure que hi ha un nombre considerable de sistemes de codificació pels fitxers de text (molts d'ells es diferencien únicament a la representació dels caràcters accentuats i especials). El propòsit d'aquest document és entendre que quan treballem amb fitxers de text des de **java** hem de tenir en compte la seva codificació.

ELS FITXERS BINARIS

Hem vist que son els fitxers de text. I els fitxers binaris?. Doncs son fitxers que poden tenir qualsevol contingut. Els fitxers executables (**.EXE**, **.COM**, **.DLL**) son un clar exemple de fitxers binaris. Si faig un **type** d'un fitxer binari poden aparèixer codis estranys que no seran adequadament representats. Evidentment, com un fitxer binari pot tenir qualsevol contingut, els fitxers de text, son un subconjunt dels fitxers binaris, perquè estan inclosos dintre d'aquests.

Per exemple, podem comprovar-ho amb el fitxer executable **ST6UNST.EXE**.



Representació del final de línia - retorn de carro als fitxers de text

Un comentari especial requereix la representació del caràcter o caràcters que indiquen la finalització d'una línia o paràgraf en un fitxer de text.

Codificació del retorno de carro en diferents màquines i sistemes operatius al llarg de la història: (Font: https://es.qwe.wiki/wiki/Newline)

Sistema operativo	Codificación de caracteres	Abreviatura	hexagonal valor	diciembre valor	Secuencia de escape
Multics , Unix y Unix-como sistemas (Linux , MacOS , FreeBSD , AIX , Xenix , etc.), BeOS , Amiga , RISC OS , y otros	ASCII	LF	0A	10	\norte
Atari TOS , Microsoft Windows , DOS (MS-DOS , PC-DOS , etc.), diciembre TOPS-10 , RT-11 , CP / M , MP / M , OS / 2 , el sistema operativo Symbian , Palm OS , Amstrad CPC , y la mayoría de los primeros sistemas operativos no Unix y no son de IBM		CR LF	0D 0A	13 10	\ R \ n
Commodore máquinas de 8 bits (C64 , C128), Acorn BBC , ZX Spectrum , TRS-80 , de la familia Apple II , Oberon , el Mac OS clásico , MIT Lisp Máquinas y OS-9		CR	0D	13	\ r
QNX aplicación pre-POSIX (versión <4)		RS	1E	30	
Acorn BBC y RISC OS en cola de salida de texto.		LF + CR	0D 0A	10 13	\ N \ r
Atari máquinas de 8 bits	atascii		9B	155	
IBM sistemas mainframe, incluyendo z / OS (OS / 390) y i5 / OS (OS / 400)	EBCDIC	NL	15	21	\ 025
ZX80 y ZX81 (equipos domésticos de Sinclair Research Ltd)	se utiliza un conjunto específico de caracteres no ASCII	NUEVA LÍNEA	76	118	

Amb tants sistemes de codificació diferent, com saber quina és la codificació d'un fitxer de text a l'hora de treballar amb ell?. En **Unicode** ho han resolt indicant el sistema de codificació del fitxer amb **dos** o **tres bytes** a l'inici d'aquest (en **UTF-8** no és necessari indicar-ho ni es recomana, tot i que es pot fer). Aquest indicador s'anomena **BOM** (**B**yte **O**rder **M**ark), més informació a: https://en.wikipedia.org/wiki/Byte_order_mark

Si consultem la Wikipèdia (https://ca.wikipedia.org/wiki/Unicode) veiem que:

Unicode defineix tres formes de codificació amb el nom *UTF* (Unicode transformation format, en català format de transformació Unicode):

- UTF-8: codificació orientada a byte amb símbols de longitud variable.
- UTF-16: codificació de 16 bits de longitud variable optimitzada per a la representació del pla bàsic multilingüe (BMP).
- UTF-32: codificació de 32 bits de longitud fixa, i la més senzilla de les tres.

Codificació del retorn de carro en **UTF-8** (**8-bit U**nicode **T**ransformation **F**ormat) i en altres sistemes de codificació **Unicode** (Font: https://unicode-table.com/es/000D/). Més informació a: https://ca.wikipedia.org/wiki/Unicode i https://es.qwe.wiki/wiki/Unicode

Codificación

Codificación	hex	dec (bytes)	dec	binary
UTF-8	0D	13	13	00001101
UTF-16BE	00 0D	0 13	13	00000000 00001101
UTF-16LE	0D 00	13 0	3328	00001101 00000000
UTF-32BE	00 00 00 0D	0 0 0 13	13	00000000 00000000 00000000 00001101
UTF-32LE	0D 00 00 00	13 0 0 0	218103808	00001101 00000000 00000000 00000000

Tanmateix, sabem que **Linux** treballa amb **Unicode** (**UTF-8**), i que fa servir el **LF = 0AH = 10** com a finalitzador de paràgrafs. L'explicació és que, l'estàndard **Unicode**, defineix una sèrie de caràcters que les aplicacions conformes han de reconèixer com a terminadors de línia (s'indiquen a la dreta), entre elles el **LF** que utilitza **Linux**.

(Font: https://es.qwe.wiki/wiki/Newline)

LF : Avance de línea, U + 000A
VT : Pestaña vertical , U + 000B
FF : Avance página , U + 000C
CR : Retorno de carro , U + 000D
CR + LF : CR (U + 000D) seguido de LF (U + 000A)
NEL : Línea siguiente, U + 0085
LS : línea de separación, U + 2028
PS : El párrafo separador, U + 2029