

Caso práctico



Ministerio de Educación y FP [CC BY-NC](#)

Ada está repasando los requisitos de la aplicación informática que están desarrollando para la clínica veterinaria.

En particular, ahora mismo se está centrando en estudiar las necesidades respecto al almacenamiento de datos. **Ada** piensa que hay ciertas partes de la aplicación que no necesitan una base de datos para guardar los datos, y sería suficiente con emplear ficheros. Por ejemplo, para guardar datos de configuración de la aplicación.

Tras repasar, se reúne con **María** y **Juan** para planificar adecuadamente el tema de los ficheros que van a usar en la aplicación, ya que es un asunto muy importante, que no deben dejar aparcado por más tiempo.

Precisamente **Antonio**, que cada vez está más entusiasmado con la idea de estudiar algún ciclo, de momento, está matriculado y cursando el módulo de Programación, y está repasando para el examen que tiene la semana que viene, uno de los temas que le "cae" es precisamente el de almacenamiento de información en ficheros.



[Ministerio de Educación y Formación Profesional](#). (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Introducción.

Cuando desarrollas programas, en la mayoría de ellos los usuarios pueden pedirle a la aplicación que realice cosas y pueda suministrarle datos con los que se quiere hacer algo. Una vez introducidos los datos y las órdenes, se espera que el programa manipule de alguna forma esos datos, para proporcionar una respuesta a lo solicitado.

Además, normalmente interesa que el programa guarde los datos que se le han introducido, de forma que si el programa termina, los datos no se pierdan y puedan ser recuperados en una sesión posterior. La forma más normal de hacer esto es mediante la utilización de ficheros, que se guardarán en un dispositivo de memoria no volátil (normalmente un disco).

Por tanto, sabemos que el almacenamiento en variables o vectores (arrays) es temporal, los datos se pierden en las variables cuando están fuera de su ámbito o cuando el programa termina. **Las computadoras utilizan ficheros para guardar los datos**, incluso después de que el programa termine su ejecución. Se suele denominar a los datos que se guardan en ficheros **datos persistentes**, porque existen, persisten más allá de la ejecución de la aplicación. Los ordenadores almacenan los ficheros en unidades de almacenamiento secundario como discos duros, discos ópticos, etc. En esta unidad veremos cómo hacer con Java estas operaciones de crear, actualizar y procesar ficheros.

A todas estas operaciones, que constituyen un flujo de información del programa con el exterior, se les conoce como **Entrada/Salida (E/S)**.

Distinguimos dos tipos de E/S: la **E/S estándar** que se realiza con el terminal del usuario y la **E/S a través de ficheros**, en la que se trabaja con ficheros de disco.

Todas las operaciones de E/S en Java vienen proporcionadas por el paquete estándar del API de Java denominado `java.io` que incorpora interfaces, clases y excepciones para acceder a todo tipo de ficheros.

El contenido de un archivo puede interpretarse como **campos** y **registros** (grupos de campos), dándole un significado al conjunto de bits que en realidad posee.



[Stephanie Booth](#) (CC BY-NC)

Para saber más

A continuación puedes ampliar tus conocimientos sobre Entrada y Salida en general, en el mundo de la informática. Verás que es un basto tema lo que abarca.

[Entrada y Salida.](#)

1.1.- Excepciones.

Cuando se trabaja con archivos, es normal que pueda haber errores, por ejemplo: podríamos intentar leer un archivo que no existe, o podríamos intentar escribir en un archivo para el que no tenemos permisos de escritura. Para manejar todos estos errores debemos utilizar excepciones. Las dos excepciones más comunes al manejar archivos son:

- ✔ **FileNotFoundException**: Si no se puede encontrar el archivo.
- ✔ **IOException**: Si no se tienen permisos de lectura o escritura o si el archivo está dañado.

Un esquema básico de uso de la captura y tratamiento de excepciones en un programa, podría ser este, importando el paquete `java.io.IOException`:

```
public static void main(String args[]) {
    try {
        // Se intenta algo que puede producir una excepción.
        int i = Integer.parseInt("1");
        // Mensaje de una excepción por no encontrar un archivo.
    } catch (IOException e) {
        // Mensaje de una excepción de entrada/salida.
    } catch (Exception e) {
        // Mensaje de una excepción cualquiera.
    } finally {
        // Siempre se ejecuta: haya o no excepción.
    }
}
```

José Javier Bermúdez Hernández. (CC BY-NC)



[Luis Fernando Pienda Mahecha](#) (CC BY)

[Código de la estructura para gestionar excepciones.](#) (1.00 KB)

Autoevaluación

Señala la opción correcta:

- Java no ofrece soporte para excepciones.
- Un campo y un archivo es lo mismo.
- Si se intenta abrir un archivo que no existe, entonces saltará una excepción.
- Ninguna es correcta.

Respuesta incorrecta, sí que las soporta.

Incorrecto, el contenido de un fichero está conformado por campos.

¡Exacto! Se disparará una excepción de tipo `FileNotFoundException`.

No has acertado, sí hay una correcta.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

Para saber más

En el siguiente enlace hay un manual muy interesante de Java, en la web **w3schools**. Puedes consultar más información sobre las excepciones en Java así como de otros aspectos del lenguaje que te puedan interesar. La parte mas interesante de este portal es que explica de manera abreviada (en inglés) los contenidos con muchos ejemplos que además pueden ser ejecutados en línea. Además, contiene numerosos ejercicios en línea que te ayudarán a afianzar todos los contenidos. Otra parte muy interesante es que también tiene contenidos de otros lenguajes, como Javascript o PHP.

[Excepciones en Java.](#)

2.- Concepto de flujo.

Caso práctico

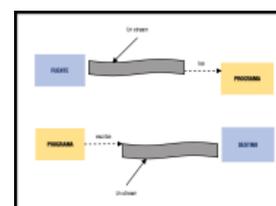
Antonio está estudiando un poco antes de irse a dormir. Se ha tomado un vaso de leche con cacao y está repasando el concepto de flujo. Entenderlo al principio, cuando lo estudió por primera vez, le costó un poco, pero ya lo entiende a la perfección y piensa que si le sale alguna pregunta en el examen de la semana que viene, sobre esto, seguro que la va a acertar.



Ministerio de Educación y FP [\(CC BY-NC\)](#)

La clase `Stream` representa un flujo o corriente de datos, es decir, un conjunto secuencial de bytes, como puede ser un archivo, un dispositivo de entrada/salida (en adelante E/S), memoria, un conector `TCP/IP` (Protocolo de Control de Transmisión/Protocolo de Internet), etc.

Cualquier programa realizado en Java que necesite llevar a cabo una operación de entrada salida lo hará a través de un **stream**.



Ministerio de Educación y FP [\(CC BY-NC\)](#)

Un flujo o stream es una abstracción de aquello que produzca o consuma información. Es una entidad lógica.

Las clases y métodos de E/S que necesitamos emplear son las mismas **independientemente del dispositivo con el que estemos actuando**, luego, el núcleo de Java, sabrá si tiene que tratar con el teclado, el monitor, un sistema de archivos o un socket de red liberando al programador de tener que saber con quién está interactuando.

La vinculación de un flujo al dispositivo físico la hace el sistema de entrada y salida de Java.

En resumen, será el flujo el que tenga que comunicarse con el sistema operativo concreto y "entendérselas" con él. De esta manera, **no tenemos que cambiar absolutamente nada en nuestra aplicación**, que va a ser independiente tanto de los dispositivos físicos de almacenamiento como del sistema operativo sobre el que se ejecuta. Esto es primordial en un lenguaje multiplataforma y tan altamente portable como Java.

Autoevaluación

Señala la opción correcta:

- La clase `Stream` puede representar, al instanciarse, a un archivo.
- Si programamos en Java, hay que tener en cuenta el sistema operativo cuando tratemos con flujos, pues varía su tratamiento debido a la diferencia de plataformas.
- La clase `keyboard` es la clase a utilizar al leer flujos de teclado.
- La vinculación de un flujo al dispositivo físico la hace el hardware de la máquina.

¡Exacto!

¡No! Es indiferente.

¡Incorrecto! Es la clase `Stream`.

¡Incorrecto! Lo hace el sistema de E/S de Java.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto
4. Incorrecto

Para saber más

En la siguiente presentación puedes aprender más sobre sockets en Java.

[Sockets en Java.](#)

[Resumen textual alternativo](#)

3.- Clases relativas a flujos.

Caso práctico

Otro aspecto importante que **Ada** trata con **María** y **Juan**, acerca de los ficheros para la aplicación de la clínica, es el tipo de ficheros a usar. Es decir, deben estudiar si es conveniente utilizar ficheros para almacenar datos en ficheros de texto, o si deben utilizar ficheros binarios. María comenta -Quizás debemos usar los dos tipos de ficheros, dependerá de qué se vaya a guardar, -Juan le contesta -tienes razón María, pero debemos pensar entonces cómo va el programa a leer y a escribir la información, tendremos que utilizar las clases Java adecuadas según los ficheros que decidamos usar.



Ministerio de Educación y FP [\(CC BY-NC\)](#)

Existen dos tipos de flujos, flujos de bytes (byte streams) y flujos de caracteres (character streams).

- ✓ Los **flujos de caracteres** (16 bits) se usan para manipular datos legibles por humanos (por ejemplo un fichero de texto). Vienen determinados por dos clases abstractas: **Reader** y **Writer**. Dichas clases manejan flujos de caracteres **Unicode**. De ellas derivan subclases concretas que implementan los métodos definidos destacados los métodos `read()` y `write()` que, en este caso, leen y escriben **caracteres** de datos respectivamente.
- ✓ Los **flujos de bytes** (8 bits) se usan para manipular datos binarios, legibles solo por la maquina (por ejemplo un fichero de programa). Su uso está orientado a la lectura y escritura de datos binarios. El tratamiento del flujo de bytes viene determinado por dos clases abstractas que son **InputStream** y **OutputStream**. Estas dos clases son las que definen los métodos que sus subclases tendrán implementados y, de entre todos, destacan `read()` y `write()` que leen y escriben bytes de datos respectivamente.



[O'Reilly & Associates](#) (Todos los derechos reservados)

Las clases del paquete `java.io` se pueden ver en la ilustración. Destacamos las clases relativas a flujos:

- ✓ **BufferedInputStream**: permite leer datos a través de un flujo con un buffer intermedio.
- ✓ **BufferedOutputStream**: implementa los métodos para escribir en un flujo a través de un buffer.
- ✓ **FileInputStream**: permite leer bytes de un fichero.
- ✓ **FileOutputStream**: permite escribir bytes en un fichero o descriptor.
- ✓ **StreamTokenizer**: esta clase recibe un flujo de entrada, lo analiza (parse) y divide en diversos pedazos (tokens), permitiendo leer uno en cada momento.
- ✓ **StringReader**: es un flujo de caracteres cuya fuente es una cadena de caracteres o string.
- ✓ **StringWriter**: es un flujo de caracteres cuya salida es un buffer de cadena de caracteres, que puede utilizarse para construir un string.

Destacar que hay clases que se "**montan**" sobre otros flujos para modificar la forma de trabajar con ellos. Por ejemplo, con **BufferedInputStream** podemos añadir un buffer a un flujo **FileInputStream**, de manera que se mejore la eficiencia de los accesos a los dispositivos en los que se almacena el fichero con el que conecta el flujo.

Debes conocer

En el siguiente vídeo puedes clarificar algunos de estos conceptos.

<https://www.youtube.com/embed/-1C-wVnCd3c>

[Resumen textual alternativo](#)

3.1.- Ejemplo comentado de una clase con flujos.

Vamos a ver un ejemplo con una de las clases comentadas, en concreto, con `StreamTokenizer`.

La clase `StreamTokenizer` obtiene un flujo de entrada y lo divide en "tokens". El flujo tokenizer puede reconocer identificadores, números y otras cadenas.

El ejemplo que puedes descargar en el siguiente recurso, muestra cómo utilizar la clase `StreamTokenizer` para contar números y palabras de un fichero de texto. Se abre el flujo con ayuda de la clase `FileReader`, y puedes ver cómo se "monta" el flujo `StreamTokenizer` sobre el `FileReader`, es decir, que se construye el objeto **`StreamTokenizer`** con el flujo `FileReader` como argumento, y entonces se empieza a iterar sobre él.



[JD Hancock \(CC BY\)](#)

[Clase para leer palabras y números.](#)

El método `nextToken` devuelve un `int` que indica el tipo de token leído. Hay una serie de constantes definidas para determinar el tipo de token:

- ✔ `TT_WORD` indica que el token es una palabra.
- ✔ `TT_NUMBER` indica que el token es un número.
- ✔ `TT_EOL` indica que se ha leído el fin de línea.
- ✔ `TT_EOF` indica que se ha llegado al fin del flujo de entrada.

En el código de la clase, apreciamos que se iterará hasta llegar al fin del fichero. Para cada token, se mira su tipo, y según el tipo se incrementa el contador de palabras o de números.

Autoevaluación

Indica si la siguiente afirmación es verdadera o falsa.

Según el sistema operativo que utilicemos, habrá que utilizar un flujo u otro. ¿Verdadero o Falso?

Verdadero Falso

Verdadero

El sistema operativo es indiferente, el flujo se encargará de los detalles subyacentes en la comunicación, por lo que el programador no tiene que preocuparse de esas cuestiones.

4.- Flujos.

Caso práctico



Ministerio de Educación y FP (CC BY-NC)

Ana y **Antonio** salen de clase. Antonio ha quedado con una amiga y Ana va camino de casa pensando en lo que le explicaron en clase hace unos días. Como se quedó con dudas, también le consultó a **María**. En concreto, le asaltaban dudas sobre cómo leer y escribir datos por teclado en un programa, y también varias dudas sobre lectura y escritura de información en ficheros. **María** le solventó las dudas hablándole sobre el tema, pero aún así, tenía que probarlo tranquilamente en casa, haciéndose unos pequeños ejemplos, para comprobar toda la nueva información aprendida.

-Antes de irte, -dice Antonio a Ana, -siéntate a hablar con nosotros un rato.

-Bueno, pero me voy a ir enseguida, -contesta Ana-

Hemos visto qué es un flujo y que existe un árbol de clases amplio para su manejo. Ahora vamos a ver en primer lugar los **flujos predefinidos**, también conocidos como de entrada y salida, y después veremos los **flujos basados en bytes** y los **flujos basados en carácter**.



David.nikonvscanon (CC BY)

Citas para pensar

"Lo escuché y lo olvidé, lo vi y lo entendí, lo hice y lo aprendí". Confucio.

4.1.- Flujos predefinidos. Entrada y salida estándar.

Tradicionalmente, los usuarios del sistema operativo Unix, Linux y también MS-DOS, han utilizado un tipo de entrada/salida conocida comúnmente por entrada/salida estándar. El fichero de entrada estándar (`stdin`) es típicamente el teclado. El fichero de salida estándar (`stdout`) es típicamente la pantalla (o la ventana del terminal). El fichero de salida de error estándar (`stderr`) también se dirige normalmente a la pantalla, pero se implementa como otro fichero de forma que se pueda distinguir entre la salida normal y (si es necesario) los mensajes de error.



[Ces-VLC \(CC BY-NC\)](#)

Java tiene acceso a la entrada/salida estándar a través de la clase `System`. En concreto, los tres ficheros que se implementan son:

- ✓ `Stdin`. Es un objeto de tipo `InputStream`, y está definido en la clase `System` como flujo de entrada estándar. Por defecto es el teclado, pero puede redirigirse para cada host o cada usuario, de forma que se corresponda con cualquier otro dispositivo de entrada.
- ✓ `Stdout`. `System.out` implementa `stdout` como una instancia de la clase `PrintStream`. Se pueden utilizar los métodos `print()` y `println()` con cualquier tipo básico Java como argumento.
- ✓ `Stderr`. Es un objeto de tipo `PrintStream`. Es un flujo de salida definido en la clase `System` y representa la salida de error estándar. Por defecto, es el monitor, aunque es posible redireccionarlo a otro dispositivo de salida.

Para la entrada, se usa el método `read` para leer de la entrada estándar:

- ✓ `int System.in.read();`
 - ✦ Lee el siguiente `byte` (`char`) de la entrada estándar.
- ✓ `int System.in.read(byte[] b);`
 - ✦ Leer un conjunto de bytes de la entrada estándar y lo almacena en el vector `b`.

Para la salida, se usa el método `print` para escribir en la salida estándar:

- ✓ `System.out.print(String);`
 - ✦ Muestra el texto en la consola.
- ✓ `System.out.println(String);`
 - ✦ Muestra el texto en la consola y seguidamente efectúa un salto de línea.

Normalmente, para **leer valores numéricos**, lo que se hace es tomar el valor de la entrada estándar en forma de cadena y entonces usar métodos que permiten transformar el texto a números (`int`, `float`, `double`, etc.) según se requiera.

Funciones de conversión.

| Método | Funcionamiento |
|---|---|
| <code>byte Byte.parseByte(String)</code> | Convierte una cadena en un número entero de un byte |
| <code>short Short.parseShort(String)</code> | Convierte una cadena en un número entero corto |
| <code>int Integer.parseInt(String)</code> | Convierte una cadena en un número entero |
| <code>long Long.parseLong(String)</code> | Convierte una cadena en un número entero largo |
| <code>float Float.parseFloat(String)</code> | Convierte una cadena en un número real simple |
| <code>double Double.parseDouble(String)</code> | Convierte una cadena en un número real doble |
| <code>boolean Boolean.parseBoolean(String)</code> | Convierte una cadena en un valor lógico |

4.2.- Flujos predefinidos. Entrada y salida estándar. Ejemplo.

Veamos un ejemplo en el que se lee por teclado hasta pulsar la tecla de retorno, en ese momento el programa acabará imprimiendo por la salida estándar la cadena leída.

Para ir construyendo la cadena con los caracteres leídos podríamos usar la clase `StringBuffer` o la `StringBuilder`. La clase `StringBuffer` permite almacenar cadenas que cambiarán en la ejecución del programa. `StringBuilder` es similar, pero no es síncrona. De este modo, para la mayoría de las aplicaciones, donde se ejecuta un solo hilo, supone una mejora de rendimiento sobre `StringBuffer`.

El proceso de lectura ha de estar en un bloque `try..catch`.

```
import java.io.IOException;

public class LeerTeclado {
    public static void main(String[] args) {
        // Creamos un flujo de caracteres que se ejecuta
        StringBuffer str = new StringBuffer();
        char c;
        // Por el hecho de que se ejecuta dentro de un try-catch
        try {
            // Esperamos la entrada de teclado en un bucle
            while (c = System.in.read()) != '\n') {
                // Almacena el caracter leído en la cadena str
                str.append(c);
            }
        } catch (IOException e) {
            System.out.println("Error leyendo()");
        }
        // Mostramos la cadena que se ha ido leyendo
        System.out.println("Cadena introducida: " + str);
    }
}
```

José Javier Bermúdez Hernández (CC BY-NC)

[Código del proceso de lectura.](#)

Autoevaluación

Señala la opción correcta:

- Read es una clase de System que permite leer caracteres.
- `StringBuffer` permite leer y `StringBuilder` escribir en la salida estándar.
- La clase `keyboard` también permite leer flujos de teclado.
- `Stderr` por defecto dirige al monitor pero se puede direccional a otro dispositivo.

¡Incorrecto!

¡No es correcto!

¡No! Inténtalo de nuevo

¡Bien hecho! Esa es la respuesta correcta.

Solución

1. Incorrecto
2. Incorrecto

3. Incorrecto
4. Opción correcta

Debes conocer

En este vídeo puedes ver un ejemplo sencillo y explicado sobre la gestión de las excepciones de I/O en Java.

<https://www.youtube.com/embed/2gWTVxe31g8>

[Resumen textual alternativo](#)

4.4.- Flujos basados en caracteres.

Las clases orientadas al flujo de bytes nos proporcionan la suficiente funcionalidad para realizar cualquier tipo de operación de entrada o salida, pero no pueden trabajar directamente con **caracteres Unicode**, los cuales están **representados por dos bytes**. Por eso, se consideró necesaria la creación de las clases orientadas al flujo de caracteres para ofrecernos el soporte necesario para el tratamiento de caracteres.

Para los flujos de caracteres, Java dispone de dos clases abstractas: **Reader** y **Writer**.

Reader, **Writer**, y todas sus subclases, reciben en el constructor el objeto que representa el flujo de datos para el dispositivo de entrada o salida.

Hay que recordar que **cada vez que se llama a un constructor se abre el flujo de datos y es necesario cerrarlo** cuando no lo necesitemos.

Existen muchos tipos de flujos dependiendo de la utilidad que le vayamos a dar a los datos que extraemos de los dispositivos.

Un flujo puede ser envuelto por otro flujo para tratar el flujo de datos de forma cómoda. Así, un **bufferWriter** nos permite manipular el flujo de datos como un buffer, pero si lo envolvemos en un **PrintWriter** lo podemos escribir con muchas más funcionalidades adicionales para diferentes tipos de datos.

En este ejemplo de código, se ve cómo podemos escribir la salida estándar a un fichero. Cuando se teclee la palabra "salir", se dejará de leer y entonces se saldrá del bucle de lectura.

Podemos ver cómo se usa **InputStreamReader** que es un puente de flujos de bytes a flujos de caracteres: lee bytes y los decodifica a caracteres. **BufferedReader** lee texto de un flujo de entrada de caracteres, permitiendo efectuar una lectura eficiente de caracteres, vectores y líneas.

Como vemos en el código, usamos **FileWriter** para flujos de caracteres, pues para datos binarios se utiliza **FileOutputStream**.

```
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
```



4.5.- Rutas de los ficheros.



Ministerio de Educación y FP [\(CC BY-NC\)](#)

En los ejemplos que vemos en el tema estamos usando la ruta de los ficheros tal y como se usan en MS-DOS, o Windows, es decir, por ejemplo:

```
c:\datos\Programacion\fichero.txt
```

Cuando operamos con rutas de ficheros, el carácter separador entre directorios o carpetas suele cambiar dependiendo del sistema operativo en el que se esté ejecutando el programa.

Para evitar problemas en la ejecución de los programas cuando se ejecuten en uno u otro sistema operativo, y por tanto persiguiendo que nuestras aplicaciones sean lo más portables posibles, se recomienda usar en Java: `File.separator`.

Podríamos hacer una función que al pasarle una ruta nos devolviera la adecuada según el separador del sistema actual, del siguiente modo:

```
private String makePathSeparator(String path){
    String separator = "/";

    try {
        // Si sistema es Windows
        // El File.separator es diferente a \
        separator = "\\";
        // Reemplazo todos los caracteres que coinciden con la expresión
        // regular para cambiarlos por el carácter File.separator
        return path.replaceAll("\\\\", File.separator);
    } catch (Exception e) {
        // Por el momento con Java 6.0.0 no se puede capturar la
        // excepción File.separator = separator, File.separator);
    }
}
```

José Javier Bermúdez Hernández [\(CC BY-NC\)](#)

[Código de separador de rutas.](#) (1 KB)

Autoevaluación

Indica si es verdadera o falsa la siguiente afirmación.

Cuando trabajamos con fichero en Java, no es necesario capturar las excepciones, el sistema se ocupa automáticamente de ellas. ¿Verdadero o Falso?

Verdadero Falso

Falso

En efecto hay que tratarlas adecuadamente para evitar malos funcionamientos de la aplicación

5.- Trabajando con ficheros.

Caso práctico

Juan le comenta a **María** -Tenemos que programar una copia de seguridad diaria de los datos de los ficheros de texto plano que utiliza el programa para guardar la información. -Mientras María escucha a Juan, recuerda que para copias de seguridad, siempre ha comprobado que la mejor opción es utilizar ficheros secuenciales.



Ministerio de Educación y FP [\(CC BY-NC\)](#)

¿Crees que es una buena opción la que piensa María o utilizarías otra en su lugar?

En este apartado vas a ver muchas cosas sobre los ficheros: cómo leer y escribir en ellos, aunque ya hemos visto algo sobre eso, hablaremos de las formas de acceso a los ficheros: secuencial o de manera aleatoria.

Siempre hemos de tener en cuenta que la manera de proceder con ficheros debe ser:

- ✔ **Abrir** o bien **crear** si no existe el fichero.
- ✔ **Hacer las operaciones** que necesitemos.
- ✔ **Cerrar el fichero**, para no perder la información que se haya modificado o añadido.



Ministerio de Educación y FP [\(CC BY-NC\)](#)

También es muy importante el **control de las excepciones**, para evitar que se produzcan fallos en tiempo de ejecución. Si intentamos abrir sin más un fichero, sin comprobar si existe o no, y no existe, saltará una excepción.

Para saber más

En el siguiente enlace a la wikipedia podrás ver la descripción de varias extensiones que pueden presentar los archivos.

[Extensión de un archivo.](#)

5.1.- Escritura y lectura de información en ficheros.

Acabamos de mencionar los pasos fundamentales para proceder con ficheros: abrir, operar, cerrar.

Además de esas consideraciones, debemos tener en cuenta también las clases Java a emplear, es decir, recuerda que hemos comentado que si vamos a tratar con ficheros de texto, es más eficiente emplear las clases de `Reader` `Writer`, frente a las clases de `InputStream` y `OutputStream` que están indicadas para flujos de bytes.



ITE (CC BY-NC)

Otra cosa a considerar, cuando se va a hacer uso de ficheros, es la forma de acceso al fichero que se va a utilizar, si va a ser de manera secuencial o bien aleatoria. En un fichero secuencial, **para acceder a un dato debemos recorrer todo el fichero desde el principio hasta llegar a su posición**. Sin embargo, en un fichero deacceso aleatorio podemos posicionarnos directamente en una posición del fichero, y ahí leer o escribir.

Aunque ya has visto un ejemplo que usa `BufferedReader`, insistimos aquí sobre la filosofía de estas clases, que usan la idea de un buffer.

La idea es que cuando una aplicación necesita leer datos de un fichero, tiene que estar esperando a que el disco en el que está el fichero le proporcione la información.

Un dispositivo cualquiera dememoria masiva, por muy rápido que sea, es mucho más lento que laCPU del ordenador.

Así que, es fundamental **reducir el número de accesos al fichero** a fin de mejorar la eficiencia de la aplicación, y para ello se asocia al fichero una memoria intermedia, el buffer, de modo que cuando se necesita leer un byte del archivo, en realidad se traen hasta el buffer asociado al flujo, ya que es una memoria mucho más rápida que cualquier otro dispositivo de memoria masiva.

Cualquier operación de Entrada/Salida a ficheros puede generar una `IOException`, es decir, un error de Entrada/Salida. Puede ser por ejemplo, que el fichero no exista, o que el dispositivo no funcione correctamente, o que nuestra aplicación no tenga permisos de lectura o escritura sobre el fichero en cuestión. Por eso, las sentencias que involucran operaciones sobre ficheros, deben ir en un bloque `try`.

Autoevaluación

Señala si es verdadera o es falsa la siguiente afirmación:

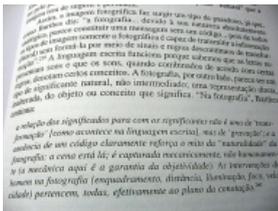
La idea de usar buffers con los ficheros es incrementar los accesos físicos a disco. ¿Verdadero o falso?

Verdadero Falso

Falso

Se trata de disminuir esos accesos.

5.2.- Archivos binarios y archivos de texto (I).



[Entropyer \(CC BY-NC\)](#)

Ya comentamos anteriormente que los ficheros se utilizan para guardar la información en un soporte: disco duro, disquetes, memorias usb, dvd, etc., y posteriormente poder recuperarla. También distinguimos dos tipos de ficheros: los de **texto** y los **binarios**.

En los **ficheros de texto** la información se guarda como caracteres. Esos caracteres están codificados en **Unicode**, o en **ASCII** u otras codificaciones de texto.

En la siguiente porción de código puedes ver cómo para un fichero existente, que en este caso es texto.txt, averiguamos la codificación que posee, usando el método `getEncoding()`

```
FileInputStream fichero;
try {
    // Creamos fichero para leer fichero de texto "archivo"
    fichero = new FileInputStream("archivo.txt");
    // Creamos un objeto de lectura de fichero de texto a caracteres
    InputStreamReader lector = new InputStreamReader(fichero);
    // Creamos un objeto de lectura de texto
    BufferedReader br = new BufferedReader(lector);
} catch (FileNotFoundException e) {
    // ...
}
```

José Javier Bermúdez Hernández [\(CC BY-NC\)](#)

[Código para mostrar codificación de fichero.](#)

Para **archivos de texto**, se puede abrir el fichero para leer usando la clase `FileReader`. Esta clase nos proporciona métodos para **leer caracteres**. Cuando nos interese no leer carácter a carácter, sino **leer líneas completas**, podemos usar la clase `BufferedReader` a partir de `FileReader`. Lo podemos hacer de la siguiente forma:

```
File arch = new File ("C:\\fich.txt");

FileReader fr = new FileReader (arch);

BufferedReader br = new BufferedReader(fr);

...

String linea = br.readLine();
```

Para **escribir en archivos de texto** lo podríamos hacer, teniendo en cuenta:

```
FileWriter fich = null;

PrintWriter pw = null;

fich = new FileWriter("/fich2.txt");

pw = new PrintWriter(fichero);

pw.println("Linea de texto");

...
```

Si el fichero al que queremos escribir existe y lo que queremos es añadir información, entonces pasaremos el segundo parámetro como `true`:

```
FileWriter("/fich2.txt",true);
```

Para saber más

En el siguiente enlace a wikipedia puedes ver el código ASCII.

[Código Ascii.](#)

5.2.1.- Ficheros binarios y ficheros de texto (II).

Los **ficheros binarios** almacenan la información en bytes, codificada en binario, pudiendo ser de cualquier tipo: fotografías, números, letras, archivos ejecutables, etc.

Los archivos binarios guardan una representación de los datos en el fichero. O sea que, cuando se guarda texto no se guarda el texto en sí, sino que se guarda su representación en código UTF-8.



[Paulnasca](#) (Dominio público)

Para **leer datos de un fichero binario**, Java proporciona la clase **FileInputStream**. Dicha clase trabaja con bytes que se leen desde el flujo asociado a un fichero. Aquí puedes ver un ejemplo comentado.

[Leer de fichero binario con buffer.](#) (3.00 KB)

Para **escribir datos a un fichero binario**, la clase nos permite usar un fichero para escritura de bytes en él, es la clase **FileOutputStream**. La filosofía es la misma que para la lectura de datos, pero ahora el flujo es en dirección contraria, desde la aplicación que hace de fuente de datos hasta el fichero, que los consume.

En la siguiente presentación puedes ver un esquema de cómo utilizar buffer para optimizar la lectura de teclado desde consola, por medio de las envolturas, podemos usar métodos como `readline()`, de la clase **BufferedReader**, que envuelve a un objeto de la clase **InputStreamReader**.



Envolturas o Wrappers

Leer desde consola

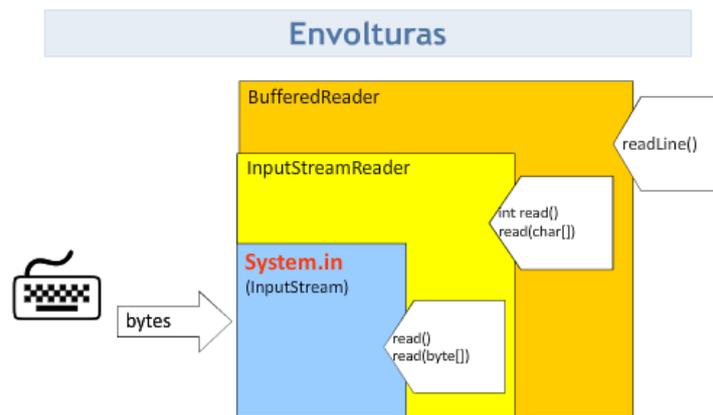
```
- BufferedReader buf = new BufferedReader(new InputStreamReader(System.in));
```

- buf es un flujo de caracteres que se enlaza a la consola a través de la clase `System.in`. Ésta se envuelve para pasar de byte a char.

- **InputStreamReader** (`InputStream inp`): clase que convierte de byte a carácter.

- **BufferedReader** (`Reader input`): clase que recibe un flujo de caracteres de entrada.

Envolturas o Wrappers



Envolturas o Wrappers

Todas las imágenes son propiedades del Ministerio de Educación y FP bajo licencia CC BY-NC.

[Resumen textual alternativo](#)

Autoevaluación

Señala si es verdadera o falsa la siguiente afirmación:

Para leer datos desde un fichero codificados en binario empleamos la clase `FileOutputStream`. ¿Verdadero o falso?

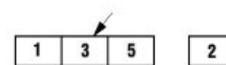
Verdadero Falso

Falso

Se trata en efecto de `FileInputStream`.

5.3.- Modos de acceso. Registros.

En Java no se impone una estructura en un fichero, por lo que conceptos como el de registro que si existen en otros lenguajes, en principio no existen en los archivos que se crean con Java. Por tanto, los programadores deben estructurar los ficheros de modo que cumplan con los requerimientos de sus aplicaciones.



Así, el programador definirá su registro con el número de bytes que le interesen, moviéndose luego por el fichero teniendo en cuenta ese tamaño que ha definido.

Ministerio de Educación y FP [\(CC BY-NC\)](#)

Se dice que un fichero es de acceso directo o de organización directa cuando para acceder a un registro n cualquiera, no se tiene que pasar por los $n-1$ registros anteriores. En caso contrario, estamos hablando de ficheros secuenciales.

Con Java se puede trabajar con **ficheros secuenciales** y con **ficheros de acceso aleatorio**.

En los **ficheros secuenciales**, la información se almacena de manera secuencial, de manera que para recuperarla se debe hacer en el mismo orden en que la información se ha introducido en el archivo. Si por ejemplo queremos leer el registro del fichero que ocupa la posición tres (en la ilustración sería el número 5), tendremos que abrir el fichero y leer los primeros tres registros, hasta que finalmente leamos el registro número tres.

Por el contrario, si se tratara de un **fichero de acceso aleatorio**, podríamos acceder directamente a la posición tres del fichero, o a la que nos interesara.

Autoevaluación

Señala la opción correcta:

- Java sólo admite el uso de ficheros aleatorios.
- Con los ficheros de acceso aleatorio se puede acceder a un registro determinado directamente.
- Los ficheros secuenciales se deben leer de tres en tres registros.
- Todas son falsas.

No es correcto, permite también los ficheros secuenciales.

¡Bien hecho! Esa es la respuesta correcta.

Incorrecto, no hay ninguna restricción de este tipo.

Respuesta incorrecta, repasa de nuevo los apuntes.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

5.4.- Acceso secuencial.



Ministerio de Educación y FP [CC BY-NC](#)

En el siguiente ejemplo vemos cómo se **escriben datos en un fichero secuencial**: el nombre y apellidos de una persona utilizando el método `writeUTF()` que proporciona `DataOutputStream`, seguido de su edad que la escribimos con el método `writeInt()` de la misma clase. A continuación escribimos lo mismo para una segunda persona y de nuevo para una tercera. Después cerramos el fichero. Y ahora lo abrimos de nuevo para ir **leyendo de manera secuencial** los datos almacenados en el fichero, y escribiéndolos a consola.

[Escribir y leer.](#)

Fíjate al ver el código, que hemos tenido la precaución de ir escribiendo las cadenas de caracteres con el mismo tamaño, de manera que sepamos luego el tamaño del registro que tenemos que leer.

Por tanto para **buscar información en un fichero secuencial**, tendremos que abrir el fichero e ir leyendo registros hasta encontrar el registro que buscamos.

¿Y si queremos **eliminar un registro en un fichero secuencial**, qué hacemos? Esta operación es un problema, puesto que no podemos quitar el registro y reordenar el resto. Una opción, aunque costosa, sería crear un nuevo fichero. Recorremos el fichero original y vamos copiando registros en el nuevo hasta llegar al registro que queremos borrar. Ese no lo copiamos al nuevo, y seguimos copiando hasta el final, el resto de registros al nuevo fichero. De este modo, obtendríamos un nuevo fichero que sería el mismo que teníamos pero sin el registro que queríamos borrar. Por tanto, si se prevé que se va a borrar en el fichero, no es recomendable usar un fichero de este tipo, o sea, secuencial.

Autoevaluación

Señala si es verdadera o es falsa la siguiente afirmación:

Para encontrar una información almacenada en la mitad de un fichero secuencial, podemos acceder directamente a esa posición sin pasar por los datos anteriores a esa información. ¿Verdadero o Falso?

Verdadero Falso

Falso

Se ha de recorrer el fichero desde el principio.

5.5.- Acceso aleatorio.

A veces no necesitamos leer un fichero de principio a fin, sino acceder al fichero como si fuera una base de datos, donde se accede a un registro concreto del fichero. Java proporciona la clase `RandomAccessFile` para este tipo de entrada/salida.

La clase `RandomAccessFile` permite utilizar un fichero de **acceso aleatorio** en el que el programador define el formato de los registros.



Ministerio de Educación y FP [\(CC BY-NC\)](#)

```
RandomAccessFile objFile = new RandomAccessFile( ruta, modo );
```

Donde ruta es la dirección física en el sistema de archivos y modo puede ser:

- ✔ "r" para sólo lectura.
- ✔ "rw" para lectura y escritura.

La clase `RandomAccessFile` implementa los interfaces `DataInput` y `DataOutput`. Para abrir un archivo en modo lectura haríamos:

```
RandomAccessFile in = new RandomAccessFile("input.dat", "r");
```

Para abrirlo en modo lectura y escritura:

```
RandomAccessFile inOut = new RandomAccessFile("input.dat", "rw");
```

Esta clase permite leer y escribir sobre el fichero, no se necesitan dos clases diferentes.

Hay que especificar el modo de acceso al construir un objeto de esta clase: sólo lectura o lectura/escritura.

Dispone de métodos específicos de desplazamiento como `seek` y `skipBytes` para poder moverse de un registro a otro del fichero, o posicionarse directamente en una posición concreta del fichero.

No está basada en el concepto de flujos o streams.

En el siguiente enlace tienes información general sobre el uso de ficheros de acceso secuencial en Java. Introduce los métodos más utilizados para el uso de este tipo de ficheros y además contiene tres ejemplos muy útiles y didácticos.

[Uso de ficheros de acceso aleatorio](#)

Autoevaluación

Indica si es verdadera o es falsa la siguiente afirmación:

Para decirle el modo de lectura y escritura a un objeto `RandomAccessFile` debemos pasar como parámetro "rw".
¿Verdadero o Falso?

Verdadero Falso

Verdadero

Esos son los parámetros para el modo lectura y escritura.

6.- Aplicaciones del almacenamiento de información en ficheros.

Caso práctico

Antonio ha quedado con **Ana** para estudiar sobre el tema de ficheros. De camino a la biblioteca, Ana le pregunta a Antonio -¿Crees que los ficheros se utilizan realmente, o ya están desfasados y sólo se utilizan las bases de datos? -Antonio tras pensarlo un momento le dice a Ana -Yo creo que sí, piensa en el mp3 que usas muchas veces, la música va grabada en ese tipo de ficheros.



Ministerio de Educación y FP [\(CC BY-NC\)](#)

¿Has pensado la **diversidad de ficheros** que existe, según la información que se guarda?

Las fotos que haces con tu cámara digital, o con el móvil, se guardan en ficheros. Así, existe una gran cantidad de ficheros de imagen, según el método que usen para guardar la información. Por ejemplo, tenemos los ficheros de extensión: .jpg, .tiff, gif, .bmp, etc.

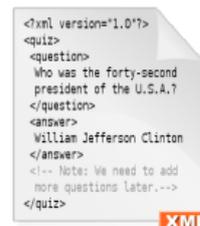
La música que oyes en tu mp3 o en el reproductor de mp3 de tu coche, está almacenada en ficheros que almacenan la información en formato mp3.

Los sistemas operativos, como Linux, Windows, etc., están constituidos por un montón de instrucciones e información que se guarda en ficheros.

El propio código fuente de los lenguajes de programación, como Java, C, etc., se guarda en ficheros de texto plano la mayoría de veces.

También se guarda en ficheros las películas en formato .avi, .mp4, etc.

Y por supuesto, se usan mucho actualmente los ficheros XML, que al fin y al cabo son ficheros de texto plano, pero que siguen una estructura determinada. XML se emplea mucho para el intercambio de información estructurada entre diferentes plataformas.



XML

Drefymac [\(CC BY-SA\)](#)

Autoevaluación

Indica si es verdadera o falsa la siguiente afirmación:

Un fichero .bmp guarda información de música codificada. ¿Verdadero o falso?

Verdadero Falso

Falso

Se trata de una imagen, mp3 sería un ejemplo de fichero para guardar música.

7.- Utilización de los sistemas de ficheros.

Caso práctico

Ana está estudiando en la biblioteca, junto a **Antonio**. Está repasando lo que le explicaron en clase sobre las operaciones relativas a ficheros en Java. En concreto, está mirando lo relativo a crear carpetas o directorios, listar directorios, borrarlos, operar en definitiva con ellos. Va a repasar ahora en la biblioteca, para tener claros los conceptos y cuando llegue de vuelta a casa, probar a compilar algunos ejemplos que a ella misma se le ocurran.

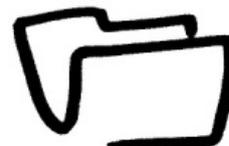


Ministerio de Educación y FP [\(CC BY-NC\)](#)

Has visto en los apartados anteriores cómo operar en ficheros: abrirlos, cerrarlos, escribir en ellos, etc.

Lo que no hemos visto es lo relativo a crear y borrar directorios, poder filtrar archivos, es decir, buscar sólo aquellos que tengan determinada característica, por ejemplo, que su extensión sea: `.txt`.

Ahora veremos cómo hacer estas cosas, y también como borrar ficheros, y crearlos, aunque crearlos ya lo hemos visto en algunos ejemplos anteriores.



[Tim Morgan \(CC BY\)](#)

Para saber más

Accediendo a este enlace, tendrás una visión detallada sobre la organización de ficheros.

[Organización de Ficheros y Métodos de Enlace](#)

7.1.- Clase File.

La clase `File` proporciona una representación abstracta de ficheros y directorios.

Esta clase, permite examinar y manipular archivos y directorios, independientemente de la plataforma en la que se esté trabajando: Linux, Windows, etc.

Las instancias de la clase `File` representan nombres de archivo, no los archivos en sí mismos.

El archivo correspondiente a un nombre dado podría ser que no existiera, por ello, habrá que controlar las posibles excepciones.

Al trabajar con `File`, las rutas pueden ser:

- ✓ Relativas al directorio actual.
- ✓ Absolutas si la ruta que le pasamos como parámetro empieza por
 - ✦ La barra "/" en Unix, Linux.
 - ✦ Letra de unidad (C:, D:, etc.) en Windows.
 - ✦ UNC(universal naming convention) en windows, como por ejemplo:

```
File miFile=new File("\\\\mimaquina\\download\\prueba.txt");
```

A través del objeto `File`, un programa puede examinar los atributos del archivo, cambiar su nombre, borrarlo o cambiar sus permisos. Dado un objeto `file`, podemos hacer las siguientes operaciones con él:

- ✓ **Renombrar** el archivo, con el método `renameTo()`. El objeto `File` dejará de referirse al archivo renombrado, ya que el `String` con el nombre del archivo en el objeto `File` no cambia.
- ✓ **Borrar** el archivo, con el método `delete()`. También, con `deleteOnExit()` se borra cuando finaliza la ejecución de la máquina virtual Java.
- ✓ **Crear** un nuevo fichero con un nombre único. El método estático `createTempFile()` crea un fichero temporal y devuelve un objeto `File` que apunta a él. Es útil para crear archivos temporales, que luego se borran, asegurándonos tener un nombre de archivo no repetido.
- ✓ **Establecer** la fecha y la hora de modificación del archivo con `setLastModified()`. Por ejemplo, se podría hacer: `new File("prueba.txt").setLastModified(new Date().getTime());` para establecerle la fecha actual al fichero que se le pasa como parámetro, en este caso `prueba.txt`.
- ✓ **Crear** un directorio con el método `mkdir()`. También existe `mkdirs()`, que crea los directorios superiores si no existen.
- ✓ **Listar** el contenido de un directorio. Los métodos `list()` y `listFiles()` listan el contenido de un directorio `list()` devuelve un vector de `String` con los nombres de los archivos, `listFiles()` devuelve un vector de objetos `File`.
- ✓ **Listar** los nombres de archivo de la raíz del sistema de archivos, mediante el método estático `listRoots()`.



Vicente Villamón (CC BY-SA)

Autoevaluación

Indica si es verdadera o falsa la siguiente afirmación:

Un objeto de la clase `File` representa un fichero en sí mismo. ¿Verdadero o falso?

Verdadero Falso

Falso

En efecto, representa un nombre de archivo y no el archivo en sí mismo.

7.2.- Interface FilenameFilter.



Pau Bou (CC BY-NC-SA)

En ocasiones nos interesa ver la lista de los archivos que encajan con un determinado criterio.

Así, nos puede interesar un filtro para ver los ficheros modificados después de una fecha, o los que tienen un tamaño mayor del que indiquemos, etc.

El interface `FilenameFilter` se puede usar para crear filtros que establezcan criterios de filtrado relativos al nombre de los ficheros. Una clase que lo implemente debe definir e implementar el método:

```
boolean accept(File dir, String nombre)
```

Este método devolverá verdadero (`true`), en el caso de que el fichero cuyo nombre se indica en el parámetro `nombre` aparezca en la lista de los ficheros del directorio indicado por el parámetro `dir`.

En el siguiente ejemplo vemos cómo se listan los ficheros de la carpeta `c:\datos` que tengan la extensión `.odt`. Usamos `try` y `catch` para capturar las posibles excepciones, como que no exista dicha carpeta.

```
public class Filtro implements FilenameFilter {
    String extension;
    Filtro(String extension) {
        this.extension = extension;
    }
    @Override
    public boolean accept(File dir, String nombre) {
        return nombre.endsWith(extension);
    }
}

public static void main(String[] args) {
    try {
        File folder = File.CURRENT_DIRECTORY;
        String[] listadosArchivos =
            listadosArchivos = Files.list(new Path(folder.toPath().resolve("datos")))
                .filter(new Filtro(".odt"))
                .toArray(new Path[0]);
        System.out.println("No hay archivos que listar");
    } catch (IOException e) {
        System.out.println("Error al listar el directorio");
    }
}

try {
    System.out.println("Error al buscar en la ruta indicada");
} catch (IOException e) {
    System.out.println("Error al buscar en la ruta indicada");
}
}
```

José Javier Bermúdez Hernández. (CC BY-NC)

[Filtrar ficheros.](#) (2.00 KB)

Autoevaluación

Indica si la siguiente afirmación es verdadera o falsa:

Una clase que implemente `FilenameFilter` puede o no implementar el método `accept`. ¿Verdadero o Falso?

Verdadero Falso

Falso

En efecto, se debe implementar siempre.

7.3.- Creación y eliminación de ficheros y directorios.

Podemos **crear un fichero** del siguiente modo:

- ✓ Creamos el objeto que encapsula el fichero, por ejemplo, suponiendo que vamos a crear un fichero llamado miFichero.txt, en la carpeta C:\\prueba, haríamos:

```
File fichero = new File("c:\\prueba\\miFichero.txt");
```

- ✓ A partir del objeto `File` creamos el fichero físicamente, con la siguiente instrucción, que devuelve un boolean con valor `true` si se creó correctamente, o `false` si no se pudo crear:

```
fichero.createNewFile()
```

Para **borrar un fichero**, podemos usar la clase `File`, comprobando previamente si existe, del siguiente modo:

- ✓ Fijamos el nombre de la carpeta y del fichero con:

```
File fichero = new File("C:\\prueba", "agenda.txt");
```

- ✓ Comprobamos si existe el fichero con `exists()` y si es así lo borramos con:

```
fichero.delete();
```

Para **crear directorios**, podríamos hacer:



José Javier Bermúdez Hernández [\(CC BY-NC\)](#)

[Crear directorios.](#)

Para **borrar un directorio** con Java tenemos que borrar cada uno de los ficheros y directorios que éste contenga. Al poder almacenar otros directorios, se podría recorrer recursivamente el directorio para ir borrando todos los ficheros.

Se puede listar el contenido del directorio con:

```
File[] ficheros = directorio.listFiles();
```

y entonces poder ir borrando. Si el elemento es un directorio, lo sabemos mediante el método `isDirectory`,

8.- Almacenamiento de objetos en ficheros. Persistencia. Serialización.

Caso práctico

Ana ya tiene los conocimientos suficientes para llevar a cabo la tarea de procesar el fichero de pedidos y poder cargarlo en una estructura de datos en memoria. Con lo aprendido en esta unidad consigue la pieza que necesita para abordar la tarea. Así:

1. Para el tratamiento del fichero podrá utilizar las clases estudiadas que permiten procesar y leer un fichero de texto línea a línea.
2. Para la validación de cada línea leída decidió utilizar expresiones regulares, que ya ha probado y validado.
3. Por último, decisión que también tenía tomada, decidió utilizar varias estructuras para almacenar los pedidos: un mapa de pedidos y una lista de artículo ordenada por código de artículo.

En el siguiente enlace puedes acceder al código completo comentado desarrollada por Ana.

[Proyecto Procesar Pedidos](#)

¡Échale un vistazo al código y tratar de implementarlo y ejecutarlo!

Caso práctico

Para la aplicación de la clínica veterinaria **María** le propone a **Juan** emplear un fichero para guardar los datos de los clientes de la clínica. -Como vamos a guardar datos de la clase Cliente, tendremos que serializar los datos.



Ministerio de Educación y FP [\(CC BY-NC\)](#)

¿Qué es la **serialización**? Es un proceso por el que **un objeto se convierte en una secuencia de bytes** con la que más tarde se podrá reconstruir el valor de sus variables. Esto permite guardar un objeto en un archivo.

Para serializar un objeto:

- ✓ éste debe **implementar el interface** `java.io.Serializable`. Este interface no tiene métodos, sólo se usa para informar a la `JVM` (Java Virtual Machine) que un objeto va a ser serializado.
- ✓ Todos los objetos incluidos en él tienen que implementar el interfaz `Serializable`.



[Ballistik Coffee Boy \(CC BY\)](#)

Todos los **tipos primitivos en Java son serializables** por defecto. (Al igual que los arrays y otros muchos tipos estándar).

Para leer y escribir objetos serializables a un stream se utilizan las clases `java: ObjectInputStream` y `ObjectOutputStream`.

En el siguiente ejemplo se puede ver cómo leer un objeto serializado que se guardó antes. En este caso, se trata de un `String` serializado:

```
FileInputStream fich = new FileInputStream("str.out");  
ObjectInputStream os = new ObjectInputStream(fich);  
Object o = os.readObject();
```

Así vemos que `readObject` lee un objeto desde el flujo de entrada `fich`. Cuando se leen objetos desde un flujo, se debe tener en cuenta qué tipo de objetos se esperan en el flujo, y se han de leer en el mismo orden en que se guardaron.

Autoevaluación

Indica si es verdadera o falsa la siguiente afirmación:

Para serializar un fichero basta con implementar el interface `Serializable`. ¿Verdadero o falso?

Verdadero Falso

Falso

Debemos asegurarnos también de que todos los objetos incluidos en él implementen el interface `Serializable`.

Para saber más

En el siguiente enlace a puedes ver un poco más sobre serialización.

[Serialización en Java.](#)

8.1.- Serialización: utilidad.

Serializable

José Javier Bermúdez Hernández. (CC BY-NC)

La serialización en Java se desarrolló para utilizarse conRMI. RMI necesitaba un modo de convertir los parámetros necesarios a enviar a un objeto en una máquina remota, y también para devolver valores desde ella, en forma de flujos de bytes. Para datos primitivos es fácil, pero para objetos más complejos no tanto, y ese mecanismo es precisamente lo que proporciona la serialización.

El método `writeObject` se utiliza para guardar un objeto a través de un flujo de salida. El objeto pasado a `writeObject` debe implementar el interfaz `Serializable`.

```
FileOutputStream fisal = new FileOutputStream("cadenas.out");  
ObjectOutputStream oos = new ObjectOutputStream(fisal);  
  
oos.writeObject();
```

La serialización de objetos se emplea también en la arquitectura de componentes software JavaBean. Las clases bean se cargan en herramientas de construcción de software visual, como NetBeans. Con la paleta de diseño se puede personalizar el bean asignando fuentes, tamaños, texto y otras propiedades.

Una vez que se ha personalizado el bean, para guardarlo, se emplea la serialización: se almacena el objeto con el valor de sus campos en un fichero con extensión `.ser`, que suele emplazarse dentro de un fichero `.jar`.

Para saber más

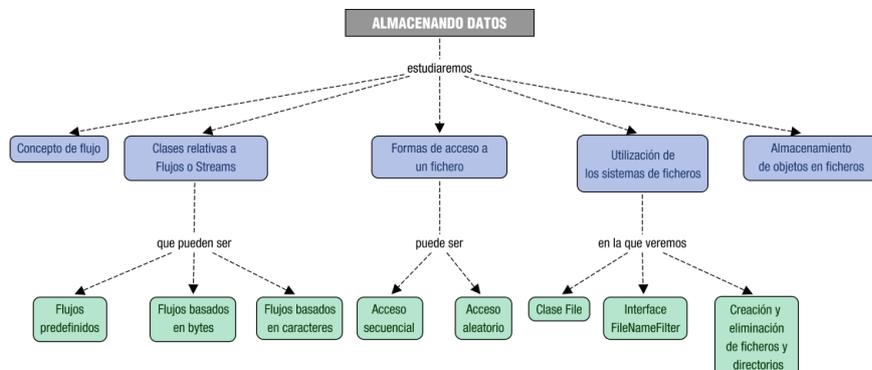
En este enlace a puedes ver un vídeo en el que se crea una aplicación sobre serialización. No está hecha con NetBeans, sino con Eclipse, pero eso no presenta ningún inconveniente.

<https://www.youtube.com/embed/4eU6WMOVmH4>

[Resumen textual alternativo](#)

9.- Conclusiones

La persistencia de datos en memoria secundaria es un requisito fundamental para las aplicaciones software: es la única forma de que los datos no se pierden cuando la aplicación finaliza la ejecución. En esta unidad hemos trabajado con todo tipo de ficheros en memoria secundaria para almacenar tanto datos binarios como caracteres. Además, hemos conocido el concepto de flujo o stream. Como hemos podido comprobar, a través del paquete `java.io`, el API Java proporciona clases e interfaces para que el almacenamiento y recuperación de datos hacia/desde ficheros sea una tarea sencilla.



Ministerio de Educación y FP [CC BY-NC](#)

Un paso más en la persistencia de datos será el uso de base de datos, algo que dejaremos para la última unidad del curso. En la siguiente unidad nos centraremos en la construcción de interfaces gráficas de usuario.