Streams, strings y codificaciones en Java

Gustavo García
20 de octubre de 2021.

Strings en Java

Toda cadena en una aplicación de Java es una instancia de la clase String. Más precisamente, hay una sola instancia de la clase String, que contiene todas las cadenas de la aplicación. Esto se llama patrón singleton[1].

4.3.3 The Class String[2]

La clase predefinida String soporta cadenas de caracteres Unicode.[3, 4] Las instancias de la clase String representan secuencias de puntos de código Unicode. Un objeto String tiene un valor constante (invariable). Los literales de cadena (§3.10.5) son referencias a instancias de la clase String. El operador de concatenación de cadenas + (§15.18.1) crea implícitamente un nuevo objeto String cuando el resultado no es una expresión constante (§15.28).
La codificación que nominalmente es Unicode, termina siendo UTF-8 a todos los efectos prácticos.

3.3 Primer paso de la compilación: Unicode Escapes[5]

Un compilador para el lenguaje de programación Java ("compilador Java") primero reconoce los escapes Unicode en su entrada, traduciendo los caracteres ASCII \u seguidos de cuatro dígitos hexadecimales a la unidad de código UTF-16 (§3.1) para el valor hexadecimal indicado, y pasando todos los demás caracteres sin cambios. La representación de caracteres suplementarios requiere dos escapes Unicode consecutivos. Este paso de traducción da como resultado una secuencia de caracteres de entrada Unicode

3.4 Segundo paso de la compilación: Line Terminators[6]

A continuación, un compilador de Java divide la secuencia de caracteres de entrada Unicode en líneas, reconociendo los terminadores de línea.

Terminador de línea:
the ASCII LF character, also known as "newline"
the ASCII CR character, also known as "return"
the ASCII CR character followed by the ASCII LF character

Carácter de entrada:
UnicodeInputCharacter salvo CR o LF

Las líneas son terminadas por los ASCII characters CR, or LF, or CR LF. Los dos caracteres CR seguidos inmediatamente por LF se cuentan como un terminador de línea, no dos.
Un terminador de línea especifica la terminación de un comentario de la forma // (§3.7).
Las líneas definidas por terminadores de línea pueden determinar los números de línea producidos por un compilador Java.
El resultado es una secuencia de terminadores de línea y caracteres de entrada, que son los símbolos de terminal para el tercer paso en el proceso de tokenización.

3.10.5 String Literals[7]

Un literal de cadena consta de cero o más caracteres entre comillas dobles.
Los caracteres se pueden representar mediante secuencias de escape (§3.10.6): una secuencia de escape para caracteres en el rango U+0000 a U+FFFF, dos secuencias de

---

[1] https://es.wikipedia.org/wiki/Singleton

[2] Java SE 14 Edition, p. 58.
[3] Java SE 14 Edition, sección 1.1, p. 2.
[4] https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/String.html

[5] Java SE 12 Edition, p. 17.
[6] Java SE 12 Edition, p. 19.
[7] Java SE 12 Edition, p. 36.

escape para las unidades de código sustituto UTF-16 de caracteres en el rango U+010000 a U+10FFFF.

StringLiteral:

"*{StringCharacter}*"

StringCharacter:

InputCharacter but not " or \
EscapeSequence

Consulte §3.10.6 para conocer la definición de *EscapeSequence*.
Un literal de cadena es siempre de tipo String (§4.3.3).
Es un error en tiempo de compilación que aparezca un terminador de línea después de la apertura " y antes del correspondiente cierre ".
Como se especifica en §3.4, los caracteres CR y LF nunca son un InputCharacter; cada uno es reconocido como un LineTerminator.
Un literal de cadena larga siempre se puede dividir en partes más cortas y escribir como una expresión (posiblemente entre paréntesis) usando el operador de concatenación de cadenas + (§15.18.1).
Los siguientes son ejemplos de literales de cadena:

```
"" // the empty string
"\"" // a string containing " alone
"This is a string" // a string containing 16 characters
"This is a " + // actually a string-valued constant expression,
"two-line string" // formed from two string literals
```

CR y LF dentro de una String

Debido a que los escapes Unicode se procesan muy pronto, no es correcto escribir "\u000a" para un literal de cadena que contiene un salto de línea (LF); el escape Unicode \u000a se transforma en un salto de línea real en el paso de traducción 1 (§3.3) y el salto de línea se convierte en un LineTerminator en el paso 2 (§3.4), por lo que el literal de cadena no es válido en el paso 3. En su lugar, se debe escribir "\n" (§3.10.6). De manera similar, no es correcto escribir "\u000d" para un literal de cadena que contiene un retorno de carro (CR). En su lugar, use "\r". Finalmente, no es posible escribir "\u0022" para un literal de cadena que contenga comillas dobles (").
Un literal de cadena es una referencia a una instancia de la clase String (§4.3.1, §4.3.3).
Además, un literal de cadena siempre se refiere a la misma instancia de la clase String. Esto se debe a que los literales de cadena - o, más generalmente, las cadenas que son los valores de expresiones constantes (§15.28) - están "internados" para compartir instancias únicas, utilizando el método String.intern (§12.5).
Los literales de cadena están representados por objetos String (§4.3.3)[8].

Establecer UTF-8 en NetBeans

Establecer UTF-8 como la codificación por defecto para NetBeans.

Si se usa Ant

Ir al directorio donde instalamos NetBeans y abrir la carpeta etc. Por ejemplo:

```
C:\Program Files\NetBeans-12.0\netbeans\etc
```

Editar el archivo

```
netbeans.conf
```

Es un archivo de texto plano. Buscar netbeans_default_options=. Está casi al final. Verificar que entre las opciones esté la siguiente:
-J-Dfile.encoding=UTF-8

[8] Java SE 12 Edition, p. 41.

El parámetro en una cadena, de modo que lo anterior hay que insertarlo, teniendo cuidado de respetar las comillas que marcan el final de la cadena.

Si se usa Maven

Hay que setear una variable de sistema.
Panel de control
> Sistema
> Opciones avanzadas de sistema
> Variables de entorno
> Variables de sistema
Buscar JAVA_TOOL_OPTIONS.
Si no existe, agregarla:
Nombre de la variable JAVA_TOOL_OPTIONS
Valor de la variable -Dfile.encoding=UTF8

Acentos, teclado, consola, Windows, Java

Encoding por defecto de Windows

La codificación por defecto en Windows 10 es UTF-16. El siguiente artículo lo explica:
https://docs.microsoft.com/en-us/windows/win32/intl/character-sets
El original está solo en inglés. La traducción es la siguiente:
Un "conjunto de caracteres" es una asignación de caracteres a sus valores de código de identificación. El conjunto de caracteres más utilizado en las computadoras hoy en día es Unicode, un estándar global para la codificación de caracteres. ***Internamente, las aplicaciones de Windows utilizan la implementación UTF-16 de Unicode.*** En UTF-16, la mayoría de los caracteres se identifican mediante códigos de dos bytes. Los caracteres suplementarios menos utilizados se representan cada uno por un "par sustituto", que es un par de códigos de dos bytes.
Intercalo el contenido del siguiente artículo:
https://docs.microsoft.com/es-es/windows/win32/intl/surrogates-and-supplementary-characters
Las aplicaciones de Windows normalmente usan UTF-16 para representar datos de caracteres Unicode. El uso de 16 bits permite la representación directa de 65.536 caracteres únicos, pero este Plano Multilingüe Básico (BMP) no es suficiente para cubrir todos los símbolos utilizados en los idiomas humanos. La versión 4.1 de Unicode incluye más de 97.000 caracteres, con más de 70.000 caracteres solo para el chino. El estándar Unicode ha establecido 16 "planos" adicionales de caracteres, cada uno del mismo tamaño que el BMP. Naturalmente, la mayoría de los puntos de código más allá del BMP aún no tienen caracteres asignados, pero la definición de los planos le da a Unicode la posibilidad de definir 1.114.112 caracteres (es decir, 2^{16} * 17 caracteres) dentro del rango de puntos de código U+0000 a U+10FFFF. Para que UTF-16 represente este conjunto mayor de caracteres, el estándar de Unicode define "caracteres suplementarios".
Un carácter complementario es un carácter ubicado más allá del BMP, y un "sustituto" es un valor de código UTF-16. Para UTF-16, se requiere un "par sustituto" para representar un solo carácter suplementario. El primer (alto) sustituto es un valor de código de 16 bits en el rango U+D800 a U+DBFF. El segundo sustituto (bajo) es un valor de código de 16 bits en el rango U+DC00 a U+DFFF. Usando el mecanismo de sustitución, UTF-16 puede admitir todos los 1.114.112 posibles caracteres Unicode. Para obtener más detalles sobre los caracteres complementarios, los sustitutos y los pares sustitutos, consulte el estándar de Unicode.
Vuelvo al documento anterior.
El estándar Unicode ha establecido 16 "planos" adicionales de caracteres, cada uno del mismo tamaño que el BMP. Naturalmente, la mayoría de los puntos de código más allá del BMP aún no tienen caracteres asignados, pero la definición de los planos le da a Unicode la posibilidad de definir 1,114,112 caracteres (es decir, 2^{16} * 17 caracteres) dentro del rango de puntos de código U+0000 a U+10FFFF. Para que UTF-16 represente este conjunto mayor de caracteres, el estándar de Unicode define "caracteres suplementarios".
Para chequear en PowerShell, pegar el comando:

```
[System.Text.Encoding]::Default
```

que produce la salida siguiente:

```
PS C:\WINDOWS\system32> [System.Text.Encoding]::Default


IsSingleByte      : True
BodyName          : iso-8859-1
```

```
EncodingName      : Western European (Windows)
HeaderName        : Windows-1252
WebName           : Windows-1252
WindowsCodePage   : 1252
IsBrowserDisplay  : True
IsBrowserSave     : True
IsMailNewsDisplay : True
IsMailNewsSave    : True
EncoderFallback   : System.Text.InternalEncoderBestFitFallback
DecoderFallback   : System.Text.InternalDecoderBestFitFallback
IsReadOnly        : True
CodePage          : 1252
```

El siguiente artículo de la Wikipedia:
https://es.wikipedia.org/wiki/Windows-1252
explica que **Windows-1252** o **CP-1252** es una codificación de caracteres del alfabeto latino, usada por defecto cuando unicode no se usa en los componentes oficiales de Microsoft Windows en inglés y en algunos lenguajes occidentales. Es una versión en la que el código de páginas de Windows está en los paquetes de LaTeX, el cual se refiere como *ansinew*.
Esta codificación es un superconjunto de ISO 8859-1, pero difiere de la ISO-8859-1 de IANA por el uso de caracteres no imprimibles en vez de caracteres de control en el rango 0x80 a 0x9F. Es conocido en Windows como el código de caracteres número 1252, y con el nombre de "windows-1252", aprobado por la IANA. Este código de caracteres también incluye todos los caracteres imprimibles de ISO 8859-15 (algunos mapeados a diferentes posiciones).
El siguiente artículo de Unicode muestra que Windows-1252 y Unicode son virtualmente idénticos:
https://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WindowsBestFit/bestfit1252.txt
A su vez, UTF-16 es simplemente un subconjunto de Unicode, como se explica en el siguiente artículo:
https://es.wikipedia.org/wiki/UTF-16#Descripci%C3%B3n

En UTF-16 cada punto de código entre U+0000 y U+FFFF se codifica, sin cambios, utilizando 16 bits. Este rango se corresponde con el *plano básico multilingüe* de Unicode, por lo que la gran mayoría de los caracteres de uso común se codifican con 16 bits.
Los caracteres de los planos adicionales se codifican mediante 32 bits. La codificación no se hace de forma directa, es decir, el código final no es el valor del punto de código. UTF-16 define un formato de transformación para estos casos denominado *pares subrogados*.
A la hora de valorar el espacio de almacenamiento requerido por un texto utilizando esta codificación, puede asumirse que los caracteres no incluidos en el plano básico son muy poco frecuentes y por lo tanto cada carácter utilizará 16 bits. Esta afirmación es válida también para el caso de las escrituras CJK (chino, japonés y coreano).

En resumen: Windows usa el encoding Windows-1252, que es el UTF-16, pero este encoding es en la práctica idéntico al Unicode. Windows usa UTF-16 porque cuando comenzó a dar soporte multilingüe el estándar de Unicode era el UTF-16. Después Unicode agregó más planos de caracteres, pero ya estaba todo hecho, y era mucho lío cambiarlo. Y quedó.

Caracteres españoles en el teclado y la consola de NetBeans

System.in es un stream orientado a bytes, no a caracteres. Cada vez que se aprieta una tecla, el sistema operativo lee el código de esa tecla según el layout del teclado que uno tenga. El SO le manda ese código al runtime de Java. Este, a su vez, recibe ese dato por el puerto correspondiente, y lo hace pasar por algo que conceptualmente podríamos llamar un driver. El resultado final es que una vez procesado por Java, lo que le llega a la variable (adentro del programa Java, o sea el jar) es el código UTF-8 (no UTF-16) de la tecla que apretamos. Eso sucede tanto para char como para String. El siguiente código lo demuestra. Está en la clase TecladoConsola dentro del proyecto streams en que estamos trabajando:

```
 1 /*
 2  Gustavo García 02-08-2019
 3  */
 4 package gui;
 5
 6 import java.io.IOException;
 7 import java.io.OutputStreamWriter;
 8 import java.io.PrintWriter;
 9 import java.nio.charset.StandardCharsets;
10 import java.util.Scanner;
11
12 public class TecladoConsola {
```

```
13
14      public static void main(String[] args) throws IOException {
15          /*
16          System.in es un stream orientado a bytes, no a caracteres. Esto significa
17          que por cada golpe de tecla, lee el código de esa tecla
18          según el sistema operativo y el layout del teclado que uno tenga.
19          Las siguientes líneas producen la salida:
20          Entrar la letra enie minuscula: ñ
21           */
22          System.out.println("1 - Entrar la letra enie minuscula: ");
23          int c = System.in.read();
24          System.out.println(c);
25          /*
26          Imprime 241 y un cuadrado.
27          En Windows-1252 (CP1252) el código de la ñ es 241.
28          En UTF-8 es C3B1. En UNICODE y en UTF-16 es U+00F1.
29          Pero F1 en hexadecimal es igual a 241 en decimal. O sea, es el mismo valor.
30          Las siguientes líneas producen la salida:
31          Unicode character ? integer 241 hex f1 string 241
32           */
33          System.out.printf("2 - Unicode character %c integer %d hex %h string %s%n",
c, (int) c, c, c);
34          String fmtStr = String.format("3 - Unicode character %c integer %d hex %h
string %s%n", c, (int) c, c, c);
35          System.out.println(fmtStr);
36          /*
37          Construye un nuevo Scanner que produce valores escaneados desde el flujo de
38          entrada especificado. Los bytes del flujo se convierten en caracteres
39          utilizando el conjunto de caracteres predeterminado de la plataforma
40          subyacente.
41           */
42          Scanner sc = new Scanner(System.in);
43          /*
44          Las siguientes líneas producen la salida:
45          Entrar la letra enie minuscula: ñ
46           */
47          System.out.println("4 - Entrar la letra enie minuscula: ");
48          String str = sc.next();
49          System.out.println(str);
50          /*
51          str tiene ahora una cadena de caracteres codificada en el encoding por
52          defecto del sistema, sea el que sea.
53          Las siguientes líneas producen la salida:
54          Unicode character ? integer 241 hex f1 string ?
55           */
56          fmtStr = String.format("5 - Unicode character %c integer %d hex %h string
%s%n",
57                  str.charAt(0), (int) str.charAt(0), str, str);
58          System.out.println(fmtStr);
59          /*
60          La documentación oficial del método PrintSream.print(String) dice:
61          Imprime una cadena. Si el argumento es nulo, se imprime la cadena "null".
62          De lo contrario, los caracteres de la cadena se convierten en bytes de
63          acuerdo con la codificación de caracteres predeterminada de la plataforma,
64          y estos bytes se escriben exactamente a la manera del método write(int).
65          Para que la consola me muestre correctamente la String, le tengo que
66          informar que la codifique en UTF-8.
67          Las siguientes líneas producen la salida:
68          6 - Ahora probamos con OutputStreamWrite
69          5 - Unicode character Ã± integer 241 hex f1 string Ã±
70           */
71          System.out.println("6 - Ahora probamos con OutputStreamWriter");
72          OutputStreamWriter consola = new OutputStreamWriter(System.out,
StandardCharsets.UTF_8);
73          consola.write(fmtStr);
74          consola.flush();
75          /*
```

```
76          Las siguientes líneas producen la salida:
77          7 - Ahora probamos con PrintWriter
78          5 - Unicode character Ã± integer 241 hex f1 string Ã±
79           */
80          PrintWriter prnCon = new PrintWriter(System.out, true,
StandardCharsets.UTF_8);
81          prnCon.printf("7 - Ahora probamos con PrintWriter%n");
82          prnCon.printf("%s%n", fmtStr);
83      }
84 }
85
```

El archivo pom.xml es:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.mycompany</groupId>
   <artifactId>encoding</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <maven.compiler.source>16</maven.compiler.source>
      <maven.compiler.target>16</maven.compiler.target>
   </properties>
   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>3.0.2</version>
            <configuration>
               <archive>
                  <manifest>
                     <mainClass>gui.TecladoConsola</mainClass>
                  </manifest>
               </archive>
            </configuration>
         </plugin>
      </plugins>
   </build>
</project>
```

Cuando se ejecuta el programa anterior, se obtiene en la consola de NetBeans la siguiente salida:

```
1 - Entrar la letra enie minuscula:
ñ
241
2 - Unicode character ñ integer 241 hex f1 string 241
3 - Unicode character ñ integer 241 hex f1 string 241

4 - Entrar la letra enie minuscula:
ñ
ñ
5 - Unicode character ñ integer 241 hex f1 string ñ

6 - Ahora probamos con OutputStreamWriter
5 - Unicode character Ã± integer 241 hex f1 string Ã±
7 - Ahora probamos con PrintWriter
5 - Unicode character Ã± integer 241 hex f1 string Ã±
```

El programa anterior muestra que, de cualquiera de los modos usados para leer (leer un char o un string), lo que llega es siempre 1 byte (UTF-8), no 2 bytes (UTF-16), y ese byte es un entero o char cuyo valor es el código de la tecla apretada.

Codificación interna de los caracteres en Java

En Java el tipo de dato primitivo char, la clase Character que encapsula el anterior y la clase String, usan todos internamente la codificación UTF-16, que es la versión original de UNICODE, que apareció en 1986. Esto se explica en los siguientes documentos:
https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/String.html
https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Character.html
https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Character.html#unicode
https://en.wikipedia.org/wiki/Unicode#History

Caracteres españoles en la consola de Windows

La codificación en la consola de Windows se maneja de una manera que, por decir lo menos, es compleja. Las razones son principalmente históricas. Decisiones que el momento en que se tomaron eran muy buenas, o las únicas posibles, se incorporaron de muchos modos al SO. Y quedaron. La codificación en la consola se manena por medio del comando chcp, que significa change code page. La code page por defecto en mi PC es 437. Se averigua con el comando chcp sin argumentos. En esta página de Microsoft tenemos la lista de todos los code pages:
https://docs.microsoft.com/en-us/windows/win32/intl/code-page-identifiers
Ahí vemos lo siguiente:

437	IBM437	OEM United States
1200	utf-16	Unicode UTF-16, little endian byte order (BMP of ISO 10646); available only to managed applications
1201	unicodeFFFE	Unicode UTF-16, big endian byte order; available only to managed applications
1250	windows-1250	ANSI Central European; Central European (Windows)
1252	windows-1252	ANSI Latin 1; Western European (Windows)
65001	utf-8	Unicode (UTF-8)

Cuando ejecutamos el jar del programa anterior directamente en la consola de Windows, es decir por fuera del IDE de NetBeans, probamos con varios códigos de página. Los resultados se resumen en la tabla siguiente:

Page code	System.out.print()	System.out.print()	StreamWriter
1200	Invalid code page		
1201	Invalid code page		
1250	n 110 6e 110	n 110 6e n	n 110 6e n
1252	ñ 241 f1 241	ñ 241 f1 ñ	Ã± 241 f1 Ã±
65001	0 0	0 0	0

En conclusión, si queremos escribir a la consola desde un jar, parece que una opción es cambiar el código de página a 1252, usar System.out estándar, y no usar StreamWriter.

Si en lugar de StreamWriter usamos PrintWriter, el code page 65001 funciona perfecto. En fin, es complicado.

```
cd "C:\Users\Gustavo\My Drive\TUP-ALUMNOS\LIBROS\Streams, strings y codificaciones en
Java\encoding\target"
java -jar "encoding-1.0-SNAPSHOT.jar"

1 - Entrar la letra enie minuscula:
ñ
164
2 - Unicode character ? integer 164 hex a4 string 164
3 - Unicode character ? integer 164 hex a4 string 164

4 - Entrar la letra enie minuscula:
ñ
?
5 - Unicode character ? integer 164 hex a4 string ?

6 - Ahora probamos con OutputStreamWriter
5 - Unicode character ┬ñ integer 164 hex a4 string ┬ñ
7 - Ahora probamos con PrintWriter
5 - Unicode character ┬ñ integer 164 hex a4 string ┬ñ
```

Introducción

Comenzamos discutiendo la arquitectura de Java para el manejo de archivos mediante programación. A continuación, explicamos que los datos se pueden almacenar en archivos de texto y archivos binarios, y las diferencias entre ellos. Demostramos cómo obtener información sobre archivos y directorios mediante las clases Paths y Files y las interfaces Path y DirectoryStream (paquete java.nio.file), luego analizamos cómo escribir y leer archivos.
Creamos y manipulamos archivos de texto. Podemos, entre otras cosas, guardar el estado de un objeto. Veremos que es incómodo leer datos de archivos de texto para recuperar el objeto original. Muchos lenguajes orientados a objetos (incluido Java) proporcionan formas convenientes para escribir objetos en archivos y leerlos (lo que se conoce como serialización y deserialización). Sin embargo, eso está desaconsejado.
Leer Por qué no usar la API de serialización de Java - 2020.docx.

Archivos y streams

Archivos

Es muy importante entender la diferencia entre un stream y un archivo. El stream es la información que viaja. El archivo es un destino (para un stream out) o un origen (para un stream in). El stream y el origen o destino son cosas distintas. Java puede escribir en un archivo, o leer de él. Cuando Java hace eso, de algún modo está recorriendo el archivo, y lo ve como un stream secuencial de bytes. En un sentido amplio, entonces, podemos hablar de los archivos como streams, pero estrictamente son cosas distintas.

End of file EOF

Cada sistema operativo proporciona un mecanismo para determinar el final de un archivo, como un marcador de fin de archivo o un conteo de los bytes totales en el archivo que se registra en una estructura de datos administrativos mantenidos por el sistema. Un programa Java que procesa un stream de bytes simplemente recibe una indicación del sistema operativo cuando llega al final del stream: el programa no necesita saber cómo la plataforma subyacente representa los archivos o los streams. En algunos casos, la indicación de fin de archivo se produce como una excepción. En otros, la indicación es un valor de retorno de un método invocado en un objeto de procesamiento de stream. Como ya dijimos, para escribir archivos o leer de ellos, Java usa streams. Los streams pueden ingresar y emitir datos como bytes o caracteres.

Streams

Cualquier información que viaja, para Java es un stream (flujo, en español). Un stream puede ser entrante (in) o saliente (out). La información entrante puede estar viniendo desde el teclado, desde una pistola lectora de códigos de barra o QR, desde un archivo, o incluso puede venir por LAN o WAN desde otra computadora. La información saliente puede estar viajando hacia la consola, hacia la impresora, hacia un archivo, o incluso puede estar yendo por LAN o WAN a otra computadora. Todos estos son streams para Java. Los archivos creados con streams basados en bytes son archivos binarios, mientras que los archivos creados con streams basados en caracteres son archivos de texto. Los archivos de texto pueden ser leídos por editores de texto, mientras que los archivos binarios son leídos por programas que comprenden el contenido específico del archivo y su ordenamiento. Se puede usar un valor numérico en un archivo binario para hacer cálculos, mientras que el carácter 5 es simplemente un carácter que se puede usar en una cadena de texto, como en "Mi perro Capitán tiene 5 años".

Streams basados en bytes

Los **streams basados en bytes** leen y escriben los datos en formato binario: un char es de dos bytes, un int es de cuatro bytes, un doble es de ocho bytes, etc. Las clases más importantes para los archivos binarios son:

- Paths y Path: ubicaciones / nombres de archivos, pero no su contenido.
- Files: operaciones sobre el contenido del archivo.
- El método File.toPath, que permite que el código legacy interactúe bien con la nueva API java.nio.

Además, las siguientes clases también se usan comúnmente con archivos binarios:

Input	Output
FileInputStream	FileOutputStream
BufferedInputStream	BufferedOutputStream
ByteArrayInputStream	ByteArrayOutputStream
DataInput	DataOutput

Al leer y escribir archivos binarios:

- casi siempre es una buena idea usar el almacenamiento en búfer (el tamaño predeterminado del búfer es 8K);
- a menudo es posible usar referencias a clases base abstractas, en lugar de referencias a clases concretas específicas;
- siempre es necesario prestar atención a las excepciones (en particular, IOException y FileNotFoundException).

Esto es para Java < 7:
Se debe tener cuidado con el método close():

- Por lo general, debe llamarse o los recursos provocarán memory leaks.
- Para las transmisiones que no utilizan el disco o la red, como las transmisiones ByteArrayXXX, la operación de cierre es una no-operación. En estos casos, no necesita llamar close().
- close limpiará automáticamente el stream, si es necesario.
- Al llamar a close en un stream "wrapper", se llamará automáticamente close en su stream subyacente. Llamar close() en un stream por segunda vez es una no-operación.

Clase InputStream[9]

Esta clase abstracta es la superclase de todas las clases que representan un flujo de entrada de bytes. El método read() de esta clase lee el siguiente byte de datos del stream de entrada. El byte de valor se devuelve como un int en el rango de 0 a 255. Si no hay ningún byte disponible porque se ha alcanzado el final del stream, se devuelve el valor -1. Este método se bloquea hasta que están disponibles datos de entrada, se detecta el final del stream o se lanza una excepción.

Clase OutputStream[10]

Esta clase abstracta es la superclase de todas las clases que representan un stream de salida de bytes. Un stream de salida acepta bytes de salida y los envía a algún destino. El método write(b) de esta clase escribe el byte especificado en el stream de salida. El contrato general de write(b) es que se escribe un byte en el flujo de salida. El byte que se va a escribir son los ocho bits de orden inferior del argumento b. Los 24 bits de orden superior de b se ignoran.

Clase PrintStream

La clase PrintStream[11] es una subclase de OutputStream. Un PrintStream agrega funcionalidad a otro flujo de salida, a saber, la capacidad de imprimir representaciones de varios valores de datos de manera conveniente. Todos los caracteres impresos por PrintStream se convierten en bytes usando la codificación o juego de caracteres dados, o la codificación de caracteres predeterminada de la plataforma si no se especifica. En situaciones que requieran escribir caracteres en lugar de bytes no se debe usar la clase PrintStream, sino la clase PrintWriter.

14.20.3 `try`**-with-resources**

A `try`-with-resources statement is parameterized with local variables (known

[9] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/InputStream.html
[10] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/OutputStream.html
[11] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/PrintStream.html#println()

as *resources*) that are initialized before execution of the `try` block and closed automatically, in the reverse order from which they were initialized, after execution of the `try` block. `catch` clauses and a `finally` clause are often unnecessary when resources are closed automatically.
Java SE12 LR, p. 473.

Traducir acá:
https://www.baeldung.com/java-try-with-resources

Este es de Oracle, pero para Java 8
https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html
y este para Java 7
https://docs.oracle.com/javase/7/docs/technotes/guides/language/try-with-resources.html

Traducir acá
http://www.javapractices.com/topic/TopicAction.do?Id=8

Este me parece que es demasiado
https://www.codejava.net/java-se/file-io/how-to-read-and-write-binary-files-in-java

Streams basados en caracteres

Los **streams basados en caracteres** leen y escriben los datos como una secuencia de caracteres en la cual cada carácter es de dos bytes: el número de bytes para un valor dado depende de la cantidad de caracteres en ese valor. Por ejemplo, el valor 20 requiere 4 bytes (2 caracteres a dos bytes por carácter), pero el valor 7 requiere solo dos bytes (1 carácter a dos bytes por carácter).

Codificación de caracteres

Para leer y escribir correctamente archivos de texto, debe comprender que esas operaciones de lectura y escritura siempre usan una codificación de caracteres (explícita o implícita) para traducir los bytes (los 1s y 0s) en caracteres de texto, y los caracteres en bytes. Cuando se guarda un archivo de texto, la herramienta que lo guarda siempre debe usar una codificación de caracteres (se recomienda UTF-8). Sin embargo, hay un problema. La codificación de caracteres no es, en general, explícita: no se guarda como parte del archivo en sí. Por lo tanto, un programa que consume un archivo de texto debe saber de antemano cuál es su codificación. Si no es así, lo mejor que puede hacer es hacer una suposición. Los problemas con la codificación generalmente aparecen como caracteres extraños en una herramienta que ha leído el archivo

UTF-8

Las clases basadas en caracteres necesitan usar algún encoding para convertir cada carácter a bytes. Como es lógico, por defecto usan el encoding del sistema, que en Windows no es UTF-8. Por lo tanto, para que funcionen como nosotros esperamos es necesario decirles que queremos usar UTF-8. Eso se hace del siguiente modo:

```
FileReader fr = new FileReader("entrada.txt", StandardCharsets.UTF_8);
```

Vemos que hay dos clases muy importantes para los archivos basados en caracteres:

- StandardCharsets y Charset (una clase vieja), para la codificación de los archivos de texto.
- El método File.toPath, que permite que el código legacy interactúe bien con la nueva API java.nio.

Clase Reader[12]

Clase abstracta para leer flujos de caracteres.

[12] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/Reader.html

InputStreamReader

Un InputStreamReader[13] es un puente de flujos de bytes a flujos de caracteres: lee bytes y los decodifica en caracteres usando un juego de caracteres específico. El juego de caracteres que utiliza se puede especificar por nombre o se puede dar explícitamente, o se puede aceptar el juego de caracteres predeterminado de la plataforma. Cada invocación de uno de los métodos read() de InputStreamReader puede hacer que se lean uno o más bytes del flujo de entrada de bytes subyacente. Para permitir la conversión eficiente de bytes a caracteres, se pueden leer más bytes del flujo subyacente de los necesarios para satisfacer la operación de lectura actual. Para una máxima eficiencia, considere envolver un InputStreamReader dentro de un BufferedReader.

Clase Writer[14].

Clase abstracta para escribir en flujos de caracteres.

OutputStreamWriter[15]

Un OutputStreamWriter es un puente entre los flujos de caracteres y los flujos de bytes: los caracteres escritos en él se codifican en bytes utilizando un juego de caracteres especificado. El juego de caracteres que utiliza se puede especificar por nombre o se puede dar explícitamente, o se puede aceptar el juego de caracteres predeterminado de la plataforma. Cada invocación de un método write() hace que el convertidor de codificación sea invocado en los caracteres dados. Los bytes resultantes se acumulan en un búfer antes de escribirse en el flujo de salida subyacente. Tenga en cuenta que los caracteres pasados a los métodos write() no se almacenan en búfer (solo los bytes se almacenan). Para una máxima eficiencia, considere envolver un OutputStreamWriter dentro de un BufferedWriter para evitar invocaciones frecuentes del convertidor.

PrintWriter[16]

Imprime representaciones formateadas de objetos en un flujo de salida de texto. Esta clase implementa todos los métodos print() que se encuentran en PrintStream. No contiene métodos para escribir bytes sin procesar, para lo cual un programa debería usar flujos de bytes no codificados.

Otras clases

Además, las siguientes clases también se usan comúnmente con archivos de texto, tanto para JDK 7 como para versiones anteriores:

- Scanner: permite leer archivos de forma compacta
- BufferedReader – readLine
- BufferedWriter - write + newLine

Al leer y escribir archivos de texto:

- A menudo es buena idea usar el almacenamiento en búfer (el tamaño predeterminado es 8K)
- Siempre es necesario prestar atención a las excepciones (en particular, IOException y FileNotFoundException)

Streams entrada estándar, salida estándar y error estándar

Estos tres streams son especiales, y vamos a hablar ahora de ellos. Un programa Java abre un archivo creando un objeto y asociando un stream de bytes o caracteres con él. El constructor del objeto interactúa con el sistema operativo para abrir el archivo. Java también puede asociar streams con diferentes dispositivos. Cuando un programa Java comienza a ejecutarse, crea tres objetos stream asociados con dispositivos: System.in, System.out y System.err.

- El objeto System.in (stream de entrada estándar) normalmente permite que un programa ingrese bytes desde el teclado.
- El objeto System.out (stream de salida estándar) normalmente permite que un programa envíe datos de caracteres a la pantalla.
- El objeto System.err (stream de error estándar) normalmente permite que un programa envíe mensajes de error basados en caracteres a la pantalla.

Cada stream puede ser redirigido. Para System.in, esta capacidad permite al programa leer bytes de una fuente diferente. Para System.out y System.err, permite que la salida se envíe a una ubicación

---

[13] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/InputStreamReader.html
[14] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/Writer.html
[15] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/OutputStreamWriter.html
[16] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/PrintWriter.html

diferente, como un archivo en el disco. La clase System proporciona los métodos setIn, setOut y setErr para redirigir los streams de entrada, salida y error estándar, respectivamente.

Los paquetes java.io y java.nio

Los programas en Java procesan streams con clases e interfaces del paquete java.io y los subpaquetes de java.nio, las nuevas API de E/S de Java que se introdujeron por primera vez en Java SE 6 y han sido mejoradas desde entonces. También hay otros paquetes en todas las API de Java que contienen clases e interfaces basadas en los paquetes java.io y java.nio.

E/S basada en caracteres

La entrada y salida basadas en caracteres se pueden realizar con las clases Scanner y Formatter.

- La clase Scanner puede ingresar datos desde el teclado, pero también puede leer datos de un archivo.
- La clase Formatter permite enviar datos con formato a cualquier stream basado en texto de una manera similar al método System.out.printf. Todas estas características pueden usarse para formatear archivos de texto también.

Java SE 8 agrega otro tipo de stream

Hay un tipo de flujo que se utiliza para procesar colecciones de elementos (como arrays y ArrayLists), en lugar de los streams de bytes relacionados con archivos que analizamos ahora. Este tema está fuera del alcance de este documento.

Uso de las clases e interfaces de NIO para obtener información de archivos y directorios

Las interfaces Path y DirectoryStream y las clases File y Files (todo del paquete java.nio.file) son útiles para recuperar información sobre archivos y directorios en el disco:

- **Interfaz java.nio.file.Path[17]:** los objetos de clases que implementan la interfaz Path representan la ubicación de un archivo o directorio en un sistema de archivos. Normalmente representará una ruta de archivo **dependiente** del sistema. Notar la diferencia con la clase File. Los objetos Path no abren archivos ni proporcionan ninguna capacidad de procesamiento de archivos: no tienen métodos para leer ni escribir.
- **Clase java.io.File[18]:** es una representación abstracta de los nombres de archivos y directorios. Las interfaces de usuario (es decir las ventanas gráficas con las que la gente interactúa) y los sistemas operativos utilizan strings de ruta dependientes del sistema para nombrar archivos y directorios. La clase File presenta una vista abstracta e **independiente** del sistema de las rutas jerárquicas. Notar la diferencia con la interfaz Path.
- **Clase java.nio.file.Paths:** (DEPRECATION NOTICE) Esta clase consta exclusivamente de métodos estáticos que devuelven una ruta a un archivo o directorio, a partir de una cadena de ruta o un URI, haciendo las conversiones necesarias. *Nota de la API: para obtener una ruta se recomienda usar los métodos Path.of de la interfaz Path, en lugar de los métodos get definidos en esta clase, ya que esta clase puede quedar obsoleta en una versión futura.*
- **Clase java.nio.file.Files[19]:** Esta clase consiste exclusivamente en métodos estáticos para manipulaciones comunes de archivos y directorios, como copiar, crear y eliminar archivos y directorios, obtener información sobre archivos y directorios, leer el contenido de los archivos, obtener objetos que le permiten manipular el contenido de archivos y directorios. En la mayoría de los casos, los métodos definidos aquí delegarán en el proveedor del sistema de archivos asociado las operaciones.
- **Interfaz java.nio.file.DirectoryStream<T>[20]:** los objetos de las clases que implementan esta interfaz permiten que un programa recorra el contenido de un directorio. Un objeto DirectoryStream permite usar el for mejorado para iterar sobre un directorio.

---

[17] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/nio/file/Path.html
[18] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/File.html
[19] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/nio/file/Files.html
[20] https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/nio/file/DirectoryStream.html

Vamos a dejar de lado la clase Paths, porque podría ser deprecada en un futuro. Como primer paso, obtenemos el path al archivo o directorio por medio de la interfaz Path (preferentemente) o la clase File. Y luego, hacemos las operaciones que necesitemos, usando las clases Files o DirectoryStream.

Interfaz java.nio.file.Path

Creación de objetos Path

Projecto TUP-ALUMNOS\LIBROS\Streams, strings y codificaciones en Java\streams, clase demoInterfazPath.java.

Tenemos dos opciones: a partir de una String, o a partir de un URI.
Para arrancar de una String, el método recomendado es el siguiente:

```
Path path = FileSystems.getDefault().getPath("logs", "access.log");
BufferedReader reader = Files.newBufferedReader(path, StandardCharsets.UTF_8);
```

En el ejemplo anterior, queremos que un BufferedReader lea texto de un archivo llamado "access.log". El archivo se encuentra en un directorio "logs" relativo al directorio de trabajo actual (cwd) y está codificado en UTF-8.
La clase java.nio.file.FileSystems define unos pocos métodos factory para sistemas de archivos. Esta clase define el método getDefault para obtener el sistema de archivos predeterminado y los métodos de fábrica para construir otros tipos de sistemas de archivos. El único que nos interesa por ahora es el que usamos en el ejemplo, FileSystems.getDefault(), que nos retorna el FileSystem por defecto. Notar que FileSystems y FileSystem son dos clases diferentes: hay una s de diferencia en el nombre.
El FileSystem predeterminado, obtenido invocando el método FileSystems.getDefault(), proporciona acceso al sistema de archivos que es accesible desde la máquina virtual Java. El sistema de archivos NO es el sistema operativo. Tampoco se debe confundir con NTFS. El sistema de archivos al que nos referimos acá es el árbol de directorios del disco por defecto de nuestra computadora. Si la PC es Windows, el objeto file system está implementado con la tecnología NTFS, que es propiedad de Microsoft. La clase FileSystems define métodos para crear sistemas de archivos que brindan acceso a otros tipos de sistemas de archivos (personalizados). El método FileSystem.getPath nos retorna el Path a partir de los argumentos que le pasamos.
La segunda opción, como dijimos antes, es obtener el Path a partir del URI. Este tema no lo necesitamos por ahora, y está fuera del alcance de este documento.

Rutas absolutas frente a rutas relativas

La ruta de un archivo o directorio especifica su ubicación en el disco. La ruta incluye algunos o todos los directorios que conducen al archivo o directorio. Una ruta absoluta contiene todos los directorios que conducen a un archivo o directorio específico, comenzando con el directorio raíz. Cada archivo o directorio en una unidad de disco en particular tiene el mismo directorio raíz en su ruta. Una ruta relativa es "relativa" a otro directorio, por ejemplo, una ruta relativa al directorio en el que la aplicación comenzó a ejecutarse. En Windows, el directorio raíz comienza por la letra del drive: "c:\", "d:\".

Ejemplo: Obtención de información de archivos y directorios

La clase Streams le pide al usuario que ingrese un nombre de archivo o directorio, luego usa las clases Path, Files y DirectoryStream para generar información sobre ese archivo o directorio.

```
package streams;

import java.io.IOException;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.attribute.FileTime;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;

public class Streams {

    private static final DateTimeFormatter DATE_FORMATTER_WITH_TIME
            = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss.SSS");
```

```
    public static void main(String[] args) throws IOException {
        String cwd = System.getProperty("user.dir");
        System.out.printf("Current working directory : %s%n", cwd);
        // print() y printf() no autoflushean el buffer. println() sí lo hace.
        System.out.println("Ingresar el nombre del directorio o archivo a buscar: ");
        Scanner teclado = new Scanner(System.in);
        String fileName = teclado.nextLine();
        System.out.printf("Nombre ingresado: %s", fileName);
        // create Path object based on user input
        Path path = Path.of(fileName);
        //consola.printf(path.toString());
        if (Files.exists(path)) { // if path exists, output info about it
            // display file (or directory) information
            System.out.printf("%n%s existe%n", path);
            System.out.printf("%s un directorio%n", Files.isDirectory(path) ? "Es" :
"No es");
            System.out.printf("%s un path absoluto%n", path.isAbsolute() ? "Es" : "No
es");
            DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");
            FileTime lmt = Files.getLastModifiedTime(path);
            System.out.printf("Última modificación: %s%n", fileTimeToString(lmt));
            System.out.printf("Tamaño: %s%n", Files.size(path));
            System.out.printf("Path: %s%n", path);
            System.out.printf("Path absoluto: %s%n", path.toAbsolutePath());
            if (Files.isDirectory(path)) { // output directory listing
                System.out.printf("%nContenido del directorio:%n");
                // object for iterating through a directory's contents
                DirectoryStream<Path> directoryStream = Files.newDirectoryStream(path);
                for (Path p : directoryStream) {
                    System.out.println(p);
                }
            }
        } else { // not file or directory, output error message
            System.out.printf("%n%s no existe%n", path.toString());
        }
    }

    public static String parseToString(LocalDateTime localDateTime) {
        return localDateTime.format(DATE_FORMATTER_WITH_TIME);
    }

    public static String fileTimeToString(FileTime fileTime) {
        String s = parseToString(
                fileTime.toInstant().atZone(ZoneId.systemDefault()).toLocalDateTime());
        return s;
    }

}
```

Clase Java.io.FileWriter

La clase Java FileWriter se utiliza para escribir datos orientados a caracteres en un archivo. Es una clase orientada a caracteres que se utiliza para el manejo de archivos en java. A diferencia de la clase FileOutputStream, no es necesario convertir la cadena en una matriz de bytes, ya que proporciona un método para escribir la cadena directamente.
Tiene dos constructores: uno recibe un argumento de clase String que es el nombre del archivo con su ruta, y el otro recibe como argumento un objeto de clase File.
https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/nio/file/package-summary.html

Clase Java.io.File

https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/io/File.html
La clase File es una representación abstracta del nombre y el path de archivos y directorios. Una ruta puede ser absoluta o relativa. La clase File tiene varios métodos para trabajar con directorios y

archivos, como crear nuevos directorios o archivos, eliminar y cambiar el nombre de directorios o archivos, listar el contenido de un directorio, etc.
Tiene cuatro constructores, que se diferencian por los argumentos que reciben-

Constructor	Descripción
File (File principal, String secundaria)	Crea una nueva instancia de File a partir de un nombre de ruta abstracto principal de clase File y una String de nombre secundario de acceso.
File (ruta de acceso de String)	Crea una nueva instancia de File al convertir la String de ruta de acceso dada en una ruta de acceso abstracta de clase File.
File (String principal, String secundaria)	Crea una nueva instancia de File a partir de una String de ruta de acceso principal y una String de ruta de acceso secundaria.
File (URI uri)	Crea una nueva instancia de File al convertir el File dado: URI en una ruta de acceso abstracta.

Método newBufferedWriter de la clase Files

https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/nio/file/Files.html#newBufferedWriter(java.nio.file.Path,java.nio.charset.Charset,java.nio.file.OpenOption...)

```
public static BufferedWriter newBufferedWriter(Path path, Charset cs,
OpenOption... options) throws IOException
```

Abre o crea un archivo para escribir, devolviendo un BufferedWriter que puede usarse para escribir texto en el archivo de una manera eficiente. El parámetro options especifica cómo se crea o abre el archivo. Si no hay opciones, este método funciona como si las opciones CREATE, TRUNCATE_EXISTING y WRITE estuvieran presentes. En otras palabras, abre el archivo para escribir, crear el archivo si no existe o, inicialmente, truncar un archivo regular existente a un tamaño de 0 si existe.
Los métodos Writer para escribir texto arrojan una IOException si el texto no puede codificarse utilizando el conjunto de caracteres especificado.

Parámetros

path - la ruta al archivo.
cs - el conjunto de caracteres que se usará para la codificación.
options - opciones que especifican cómo se abre el archivo, separadas por comas.

Devuelve

Un nuevo writer con búfer, con el tamaño del búfer predeterminado, para escribir texto en el archivo.

Arroja

IllegalArgumentException - si opciones contiene una combinación no válida de opciones.
IOException - si se produce un error de E / S al abrir o crear el archivo.
UnsupportedOperationException - si se especifica una opción no compatible.
SecurityException - En el caso del proveedor predeterminado, y si hay instalado un administrador de seguridad, se invoca el método checkWrite para comprobar el acceso de escritura al archivo. El método checkDelete se invoca para verificar el acceso de eliminación si el archivo se abre con la opción DELETE_ON_CLOSE.
seguir
http://www.javapractices.com/topic/TopicAction.do?Id=42

JSON

El proyecto está en
Classroom\L 2 21\PROYECTOS\file_read_write\file_read_write

Editar pom.xml y agregar esta dependencia:

```
    <dependencies>
        <!--
            Otras dependencias, cada una en su propio nodo.
        -->
        <dependency>
```

```
            <groupId>com.googlecode.json-simple</groupId>
            <artifactId>json-simple</artifactId>
            <version>1.1.1</version>
        </dependency>
    </dependencies>
```

Objeto JSON

Un objeto JSON es un texto plano, que tiene una lista de pares de la forma "propiedad":valor, separados por comas, y encerrados entre dos llaves {}. O sea:
{"propiedad1":valor1, "propiedad"2:valor2, "propiedad3":valor3, …}
El nombre de la propiedad siempre va entre dos comillas dobles. Por ejemplo: "apellido".
Si el valor de la propiedad es una cadena, se pone la cadena encerrada entre dos comillas dobles. Por ejemplo: "nombre":"José".
Si el valor de la propiedad es un objeto JSON, se pone ese objeto JSON, encerrado entre sus dos llaves correspondientes. Por ejemplo: "empleado":{"nombre":"José", "apellido":"Pérez"}.

Array de objetos JSON

Un objeto de tipo JSONArray es un array de objetos JSON. Es un texto plano que contiene una lista de objetos JSON, separados entre sí por comas, y todo encerrado entre dos corchetes []. O sea:
[ObjetoJSON1, ObjetoJSON2, ObjetoJSON3, …]

Objeto listaEmpleados

En el ejemplo, listaEmpleados es un objeto de tipo JSONArray que contiene dos objetos de tipo JSONObject. Son los dos empleados. Cada uno de esos objetos es la expresión en JSON de un empleado.

Objeto empleado

empleado es un objeto de tipo JSONObject. La clase JSONObject tiene un método put(), que agrega un par propiedad:valor al objeto sobre el que es invocado. Como ya dijimos, el nombre de la propiedad es siempre una cadena y debe ir entre dos comillas dobles. El valor, por parte, puede ser una cadena u otro objeto JSON. En el caso del objeto empleado, el nombre de la propiedad es "empleado", y el valor de esa propiedad es detallesEmpleado, que a su vez es un objeto de tipo JSONObject.

objeto detallesEmpleado

detallesEmpleado es un objeto de tipo JSONObject. Contiene los detalles del objeto empleado. Los detalles son las propiedades junto con sus correspondientes valores. En nuestro ejemplo, las propiedades son "nombre", "apellido" y "website". Estos pares de propiedad y valor se van agregando uno después de otro, con el método put().

https://howtodoinjava.com/java/library/json-simple-read-write-json-examples/