Programación

08 Lectura y Escritura de Información Externa. Ficheros

José Luis González Sánchez

Contenidos

1. Ficheros y Directorios
2. Java Nio2.
3. Expresiones regulares.
4. JSON
5. Localización

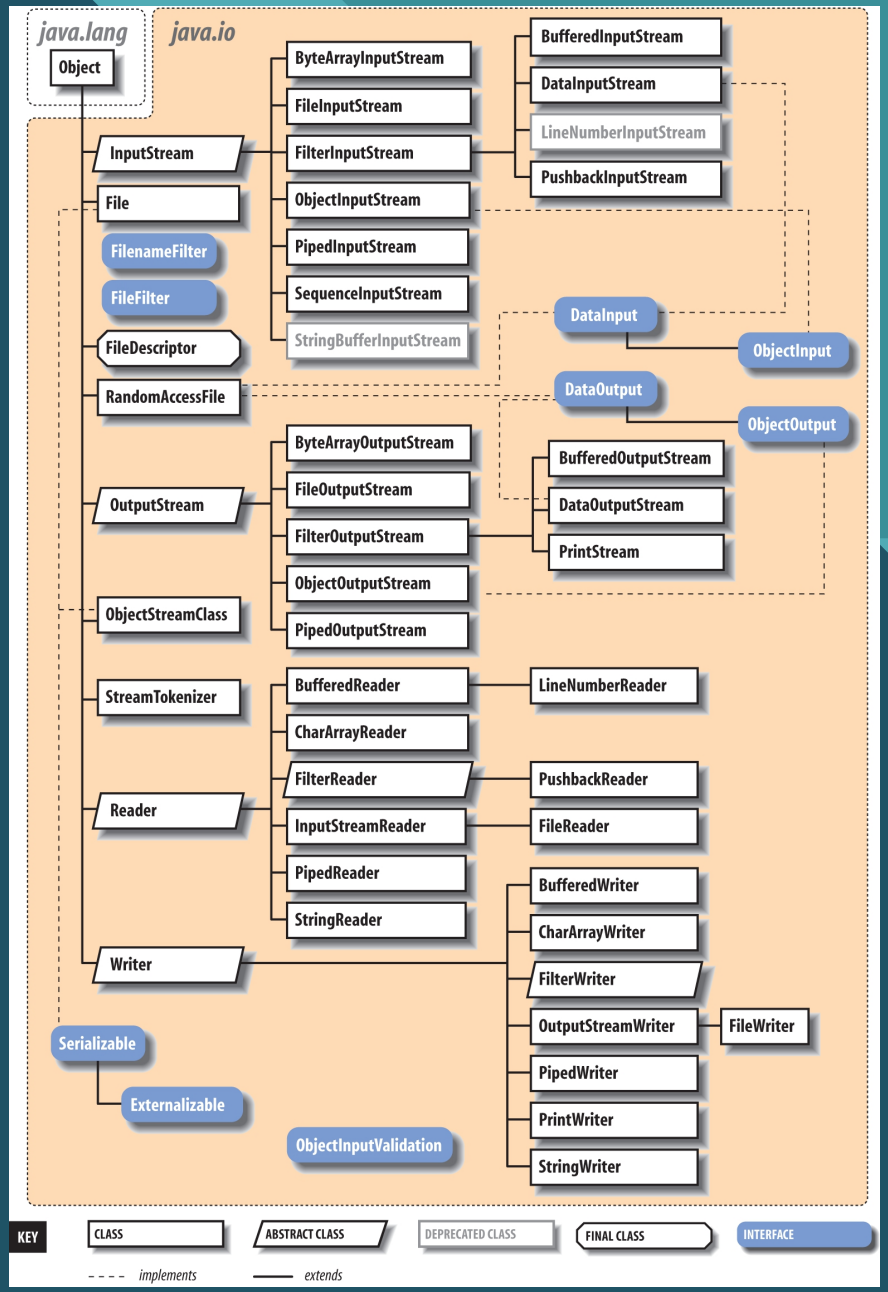
Ficheros y Directorios

Almacenamiento de la información en disco

Ficheros: Tipos y Flujos

- **Ficheros de texto** cuando el contenido del fichero contenga exclusivamente caracteres de texto (podemos leerlo con un simple editor de texto)
- **Ficheros binarios** cuando no estén compuestos exclusivamente de texto. Pueden contener imágenes, videos, ficheros, . . . aunque también podemos considerar un fichero binario a un fichero de Microsoft Word en el que sólo hayamos escrito algún texto puesto que, al almacenarse el fichero, el procesador de texto incluye alguna información binaria
- **Flujos:** son un canal de comunicación de las operaciones de entrada salida. Este esquema nos da independencia para poder trabajar igual tanto si estamos escribiendo en un fichero, como en consola, o si estamos leyendo de teclado, o de una conexión de red.
  - Flujos de entrada: sirven para introducir datos en la aplicación.
  - Flujos de salida: sirven para sacar datos de la aplicación.
  - Flujos de bytes: manejan datos en crudo.
  - Flujos de caracteres: manejan caracteres o cadenas.

Ficheros: Flujos



Ficheros: Flujos de Salida

- **Flujos de salida de bytes**. Algunas de las clases que podemos usar son:
  - OutputStream: clase abstracta, padre de la mayoría de los flujos de bytes.
  - FileOutputStream: flujo que permite escribir en un fichero, byte a byte.
  - BufferedOutputStream: flujo que permite escribir grupos (buffers) de bytes.
  - ByteArrayOutputStream: flujo que permite escribir en memoria, obteniendo lo escrito en un array de bytes.
- **Flujos hacia otros flujos**. Solo FileOutputStream tiene un constructor que acepta una ruta (entre otras opciones). El resto reciben en sus constructores un tipo de OutputStream. ¿Por qué? Porque podemos construir flujos que escriben en flujos (encadenados).
- **Flujos de salida de caracteres**
  - Writer: clase abstracta, padre de la mayoría de los flujos de caracteres.
  - FileWriter: flujo que permite escribir en un fichero, caracter a caracter.
  - BufferedWriter: flujo que permite escribir líneas de texto.
  - StringWriter: flujo que permite escribir en memoria, obteniendo lo escrito en un String
  - OutputStreamWriter: flujo que permite transformar un OutputStream en un Writer.
  - PrintWriter: flujo que permite escribir tipos básicos Java.

Ficheros: Flujos de Entrada

- **Flujos de entrada de bytes.** Algunas de las clases que podemos usar son:
  - InputStream: clase abstracta, padre de la mayoría de los flujos de bytes.
  - FileInputStream: flujo que permite leer de un fichero, byte a byte.
  - BufferedInputStream: flujo que permite leer grupos (buffers) de bytes.
  - ByteArrayInputStream: flujo que permite leer de memoria (de un array de bytes).
- **Flujos de entrada de caracteres**. Algunas de las clases que podemos usar son:
  - Reader: clase abstracta, padre de la mayoría de los flujos de caracteres.
  - FileReader: flujo que permite leer de un fichero, caracter a caracter.
  - BufferedReader: flujo que permite leer líneas de texto.
  - StringReader: flujo que permite leer desde la memoria.
  - InputStreamReader: flujo que permite transformar un InputStream en un Reader.

Ficheros: Acceso Aleatorio

- Podemos acceder de forma aleatoria a un archivo o fichero. Además, los archivos de este tipo, se pueden leer, o bien leer y escribir a la vez.
- Por otra parte, estos archivos no son stream y se numeran por un índice que empieza por cero. Este índice, se llama puntero o cursor de lectura/escritura e indica la posición a partir de la cual se empezará a leer o escribir en el archivo. Es importante indicar que la información almacenada en este tipo de archivos, se guarda en forma de bytes. Esto quiere decir que este la clase RandomAccessFile trata del archivo como un array de bytes.
- Para poder determinar cuanto ocupa cada registro en un archivo de acceso aleatorio, debemos tener en cuenta los bytes de cada uno de los datos. De esta forma, dependiendo del tipo de dato que queramos almacenar, cada registro ocupará un tamaño u otro.
- Hay que tener en cuenta que los datos tipo String (tipo texto), en java son considerados como objetos. Por este motivo, un dato tipo String, se considera un array de caracteres tipo char. Esto quiere decir que la información de tipo String, ocupará dos bytes por cada carácter tipo char. A esta información, se le deben sumar los bytes ocupados por los espacios en blanco y los saltos de línea.

Ficheros: Acceso Aleatorio

Tipo de Dato	Tamaño en Bytes
Char	2 bytes
Byte	1 byte
Short	2 bytes
Int	4 bytes
Long	8 bytes
Float	4 bytes
Double	8 bytes
Boolean	1 byte
Espacio en blanco (un char)	1 byte
Salto de línea (enter)	1 byte
String	2 bytes por cada char

Ficheros: Clase File

Nombre	Uso
isDirectory	Devuelve true si el File es un directorio
isFile	Devuelve true si el File es un fichero
createNewFile	Crea un nuevo fichero, si aun no existe.
createTempFile	Crea un nuevo fichero temporal
delete	Elimina el fichero o directorio
getName	Devuelve el nombre del fichero o directorio
getAbsolutePath	Devuelve la ruta absoluta del File
getCanonicalPath	Devuelve la ruta canónica del File
list, listFiles	Devuelve el contenido de un directorio

Java Nio.2

Más potencia y más sencillez

Java NIO.2

- En la versión 1.4 de Java se añadió un nuevo sistema de entrada/salida llamado NIO para suplir algunas de sus deficiencias que posteriormente en Java 7 se mejoró aún más con NIO.2. Entre las mejoras se incluyen permitir navegación de directorios sencillo, soporte para reconocer enlaces simbólicos, leer atributos de ficheros como permisos e información como última fecha de modificación, soporte de entrada/salida asíncrona y soporte para operaciones básicas sobre ficheros como copiar y mover ficheros.
- Las clases principales de esta nueva API para el manejo de rutas, ficheros y operaciones de entrada/salida son las siguientes:
  - **Path**: es una abstracción sobre una ruta de un sistema de ficheros. No tiene porque existir en el sistema de ficheros pero si si cuando se hacen algunas operaciones como la lectura del fichero que representa. Puede usarse como reemplazo completo de java.io.File pero si fuera necesario con los métodos File.toPath() y Path.toFile() se ofrece compatibilidad entre ambas representaciones.
  - **Files**: es una clase de utilidad con operaciones básicas sobre ficheros.
  - **FileSystems**: otra clase de utilidad como punto de entrada para obtener referencias a sistemas de archivos

Java NIO.2: Files

- **COMPROBACIONES**
  - Existencia (exists)
  - Acceso (isReadable, isWritable,isExecutable)
  - Son el mismo fichero (isSameFile)
- **COPIAR, BORRAR Y MOVER**
  - Borrar (delete, deleteIfExists)
  - Copiar (copy)
  - Mover (move)
- **CREAR, ESCRIBIR Y LEER**
  - Para crear ficheros
    - Ficheros regulares (createFile)
    - Temporales (createTempFile)
  - Buffered
    - Leer (newBufferedReader)
    - Escribir (newBufferedWriter)

Java NIO.2: Files

- **CREAR, ESCRIBIR Y LEER**
  - Unbuffered
    - Leer (newInputStream)
    - Escribir (newOutputStream)
- **TRABAJO CON DIRECTORIOS**
  - Listar
    - Raíz del sistema (FileSystem.getRootDirectories)
    - Contenido de directorios (newDirectoryStream)
  - Crear
    - Crear un directorio (createDirectory, createDirectories)
    - Temporal (createTempDirectory)

Java NIO.2: Path

- Con la clase **Path** se pueden hacer operaciones sobre rutas como obtener la ruta absoluta de un Path relativo o el Path relativo de una ruta absoluta, de cuanto elementos se compone la ruta, obtener el Path padre o una parte de una ruta.
- Otros métodos interesantes son relativize(), normalize(), toAbsolutePath(), resolve(), startsWith() y endsWith().

Expresiones Regulares

Encontrar patrones en nuestros textos

Expresiones Regulares

- Una expresión regular define un patrón de búsqueda para cadenas de caracteres.
- La podemos utilizar para comprobar si una cadena contiene o coincide con el patrón. El contenido de la cadena de caracteres puede coincidir con el patrón 0, 1 o más veces.
- Algunos ejemplos de uso de expresiones regulares pueden ser:
  - para comprobar que la fecha leída cumple el patrón dd/mm/aaaa
  - para comprobar que un NIF está formado por 8 cifras, un guión y una letra
  - para comprobar que una dirección de correo electrónico es una dirección válida.
  - para comprobar que una contraseña cumple unas determinadas condiciones.
  - Para comprobar que una URL es válida.
  - Para comprobar cuántas veces se repite dentro de la cadena una secuencia de caracteres determinada.
- El patrón se busca en el String de izquierda a derecha. Cuando se determina que un carácter cumple con el patrón este carácter ya no vuelve a intervenir en la comprobación.

Expresiones Regulares

- Cuando queremos comparar una expresión regular con una cadena de texto, podemos querer dos posibles cosas:
  - Contiene una parte de la cadena de texto que cumpla esa expresión regular. Por ejemplo, podemos querer buscar una fecha en formato "dd/mm/yyyy" en la cadena "Hoy es 12/02/2017" y efectivamente, la cadena contiene una fecha en ese formato.
  - La cadena Coincide totalmente con la expresión regular. En el ejemplo anterior, la cadena "Hoy es 12/02/2017" sí contiene una fecha en formato "dd/mm/yyyy" dentro, pero NO coincide con ese formato, puesto que le sobra el trozo de texto "Hoy es " para tener la coincidencia exacta con una fecha.
- **Símbolos comunes en expresiones regulares**
  - .: Un punto indica cualquier carácter
  - ^expresión: El símbolo ^ indica el principio del String. En este caso el String debe contener la expresión al principio.
  - expresión$: El símbolo $ indica el final del String. En este caso el String debe contener la expresión al final.
  - [abc]: Los corchetes representan una definición de conjunto. En este ejemplo el String debe contener las letras a ó b ó c.
  - [abc][12]: El String debe contener las letras a ó b ó c seguidas de 1 ó 2
  - [^abc]: El símbolo ^ dentro de los corchetes indica negación. En este caso el String debe contener cualquier carácter excepto a ó b ó c.
  - [a-z1-9]: Rango. Indica las letras minúsculas desde la a hasta la z (ambas incluidas) y los dígitos desde el 1 hasta el 9 (ambos incluidos)
  - A|B: El carácter | es un OR. A ó B
  - AB: Concatenación. A seguida de B

Expresiones Regulares

- **Meta caracteres**
  - \d: Dígito. Equivale a [0-9]
  - \D: No dígito. Equivale a [^0-9]
  - \s: Espacio en blanco. Equivale a [\t\n\x0b\r\f]
  - \S: No espacio en blanco. Equivale a [^\s]
  - \w: Una letra mayúscula o minúscula, un dígito o el carácter
  - _: Equivale a [a-zA-Z0-9_]
  - \W: Equivale a [^\w]
  - \b: Límite de una palabra.
- **Cuantificadores**
  - {X}: Indica que lo que va justo antes de las llaves se repite X veces
  - {X,Y}: Indica que lo que va justo antes de las llaves se repite mínimo X veces y máximo Y veces. También podemos poner {X,} indicando que se repite un mínimo de X veces sin límite máximo.
  - *: Indica 0 ó más veces. Equivale a {0,}
  - +: ndica 1 ó más veces. Equivale a {1,}
  - ?: Indica 0 ó 1 veces. Equivale a {0,1}

Expresiones Regulares en JAVA

- **Clase Pattern:** Un objeto de esta clase representa la expresión regular. Contiene el método compile(String regex) que recibe como parámetro la expresión regular y devuelve un objeto de la clase Pattern.
- **La clase Matcher:** Esta clase compara el String y la expresión regular. Contienen el método matches(CharSequence input) que recibe como parámetro el String a validar y devuelve true si coincide con el patrón. El método find() indica si el String contienen el patrón.

- **Utilidades:**
  - **https://regex101.com/**
  - **https://regexr.com/**
  - **https://www.regextester.com/**
  - **https://www.freeformatter.com/regex-tester.html**

JSON

Intercambiando información con JSON

JSON

- JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos.
- Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - Diciembre 1999.
- JSON es un formato de texto que es completamente independiente del lenguaje
- Estas propiedades hacen que JSONsea un lenguaje ideal para el intercambio de datos. (Fuente: http://www.json.org/json-es.html)
- JSON está constituido por dos estructuras:
  - Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un array asociativo.
  - Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arrays, vectores, listas o secuencias.
- Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

JSON

- En JSON, se presentan de estas formas:
  - Como un objeto, conjunto desordenado de pares nombre/valor. Un objeto comienza con { (llave de apertura) y termine con } (llave de cierre). Cada nombre es seguido por: (dos puntos) y los pares nombre/valor están separados por , (coma). En el ejemplo creo un objeto persona con nombre y oficio, y un objeto zona con su código y su nombre:

    { "persona": {"nombre": "Ana","oficio": "Profesora" }}

    {"zona": {"codzona": 10,"nombre": "Leganes" }}
  - Un array, es decir, una colección de valores. Un array comienza con [(corchete izquierdo) y termina con] (corchete derecho). Los valores se separan por, (coma). En el ejemplo creo el objeto persona, un array de dos elementos, no tienen por qué tener los mismos pares nombre/valor, y el objeto zona con dos zonas:

    {"persona": [

    {"nombre":"Alicia","oficio":"Profesora","ciudad": "Talavera"},

    {"nombre": "María Jesús","oficio":"Profesora"}]}

    {"zona": [

    {"codzona": 10,"nombre": "Madrid"},

    {"'codzona": 20,"nombre": "Toledo", "tasa": 15}]}

JSON

- En JAVA usaremos GSON:
  - Serializar: toJson() + pretty
  - Deserializar: fromJson()

  - Más información:
    - https://www.adictosaltrabajo.com/2012/09/17/gson-java-json/
    - https://jarroba.com/gson-json-java-ejemplos/

Localización

Adaptando la salida de datos

Localización

- En JAVA podemos usar java.util.Locale para aplicar que la salida de nuestras cadenas se adapte a nivel internacional en base a los códigos y estándares de cada país.
- https://www.oracle.com/technical-resources/articles/javase/locale.html
- https://docs.oracle.com/javase/tutorial/i18n/format/numberFormat.html
- https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html
- Redondeando a dos decimales:
  - DecimalFormat dFormat = (DecimalFormat) NumberFormat.getInstance(Locale.getDefault());
  - dFormat.setMaximumFractionDigits(2);
  - dFormat.setMinimumFractionDigits(2);
  - DecimalFormat dFormat = new DecimalFormat("#.00");
- Formateando Fechas:
  - LocalDate date = LocalDate.now();
  - DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy MM dd");
  - String text = date.format(formatter);
  - LocalDate parsedDate = LocalDate.parse(text, formatter);

Localización

- Formateando Fechas:
- https://www.logicbig.com/how-to/code-snippets/jcode-java-8-date-time-api-localdate-format.html
- https://lokalise.com/blog/java-localdate-localization/
  - LocalDate date = LocalDate.now();
  - DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy MM dd");
  - String text = date.format(formatter);
  - LocalDate parsedDate = LocalDate.parse(text, formatter);
  - LocalDate currentDate = LocalDate.now();
  - DateTimeFormatter usDateFormatter = DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL).withLocale(Locale.US);
  - String usFormattedDate = currentDate.format(usDateFormatter);
  - System.out.println("Current date in en-US date format: " + usFormattedDate);
  - DateTimeFormatter esDateFormatter = DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL).withLocale(Locale.SPAIN);
  - String esFormattedDate = currentDate.format(esDateFormatter);
  - System.out.println("Current date in es-ES date format: " + esFormattedDate);

"Menos del 10% del código tienen que ver directamente con el propósito del sistema; el resto tiene que ver con la entrada y salida, validación de datos, mantenimiento de estructuras de datos y otras labores domésticas"

- Mary shaw

Recursos

- Twitter: https://twitter.com/joseluisgonsan
- GitHub: https://github.com/joseluisgs
- Web: https://joseluisgs.github.io
- Discord: https://discord.gg/eFMKxfPY
- Aula Virtual: https://aulavirtual33.educa.madrid.org/ies.luisvives.leganes/course/view.php?id=247




Gracias

José Luis González Sánchez