CSV

El formato CSV

Un documento CSV (Comma Separated Values) es una forma almacenar datos de forma persistente. Por lo general, se emplean para la lectura de datos que no van a cambiar, ya que la escritura en CSV no es eficiente. La lectura de un CSV suele ser completa, es decir, se suele leer de forma completa de principio a fin.

> Cuando se requiere la lectura y escritura de datos aleatorios, se utilizan base de datos, que resultan óptimas para este tipo de tareas.

El formato CSV no está estandarizado, y el conjunto de símbolos utilizados para separar y escapar datos puede variar de un documento a otro. La norma técnica de 2005 RFC 4180 formaliza el formato de archivo CSV y define el tipo MIME "texto / csv" para el manejo de campos basados en texto. Sin embargo, la interpretación del texto de cada campo sigue siendo específica de la aplicación. Los archivos que siguen el estándar RFC 4180 pueden simplificar el intercambio de CSV y deben ser ampliamente portátiles. Entre sus requisitos:

- Líneas de estilo MS-DOS que terminan con caracteres (CR / LF) (opcional para la última línea).
- Un registro de encabezado opcional (no hay una forma segura de detectar si está presente, por lo que se debe tener cuidado al importar).
- Cada registro debe contener el mismo número de campos separados por comas.
- Se puede citar cualquier campo (con comillas dobles).
- Los campos que contienen un salto de línea, comillas dobles o comas deben estar entrecomillados. (Si no es así, probablemente será imposible procesar correctamente el archivo).
- Si se utilizan comillas dobles para encerrar campos, las comillas dobles deben estar representadas por dos caracteres de comillas dobles.

Un documento CSV siguiendo estas reglas puede ser el siguiente:

```
"Juan Pérez García","33","Córdoba","1996"
"Ana Gutiérrez Tolosa","21","Granada","1994"
"Pedro Rodríguez Antúnez","26","Cuenca","1997"
```

> Un documento CSV puede ser mucho más grande, con miles de líneas.

Según lo explicado anteriormente, si es necesario incluir comillas en los campos, es preciso escaparlo mediante dobles comillas. Por ejemplo, en el siguiente documento CSV, se incluye el apodo de la primera persona entre comillas dobles (escapadas):

```
"Juan ""Carapalo"" Pérez García","33","Córdoba","1996"
"Ana Gutiérrez Tolosa","21","Granada","1994"
"Pedro Rodríguez Antúnez","26","Cuenca","1997"
```

Sin embargo, CSV, al no tratarse de un formato estándar, puede utilizar diferentes símbolos. Por ejemplo:

```
Juan Pérez García;33;Córdoba;1996
Ana Gutiérrez Tolosa;21;Granada;1994
Pedro Rodríguez Antúnez;26;Cuenca;1997
```

> En este caso se utiliza *;* como símbolo separador. Este formato es menos conveniente, puesto será más difícil distinguir un símbolo *;* incluido en uno de los campos.

CSV Simple / Complejo

En este documento vamos a distinguir entre CSV simple y complejo.

- CSV Simple: no hay duda entre los símbolos. El símbolo separador sólo aparece para seperar campos, y las comillas sólo aparecen para marcar campos. Por ejemplo, el siguiente documento:

```
"Juan Pérez García","33","Córdoba","1996"
"Ana Gutiérrez Tolosa","21","Granada","1994"
"Pedro Rodríguez Antúnez","26","Cuenca","1997"
```

- CSV Complejo: los símbolos separador y de marcado de campos puede aparecer con otra finalidad. Por ejemplo, en el siguiente documento aparece el símbolo ',' como carácter dentro de los campos, y el símbolo '"' como carácter en los campos, además de para marcar campos:

```
"Pérez García, ""Torbellino"" Juan","33","Córdoba","1996"
"Gutiérrez Tolosa, ""Ana","21","Granada","1994"
"Rodríguez Antúnez, Pedro","26","Cuenca","1997"
```

Es preciso conocer de qué forma está coinstruído un CSV.

Caso 1. Lectura de un CSV simple con BufferedReader

Como ejemplo de CSV usaremos el siguiente:

```
"Juan Pérez García","33","Córdoba","1996"
"Ana Gutiérrez Tolosa","21","Granada","1994"
"Pedro Rodríguez Antúnez","26","Cuenca","1997"
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;

public class ficheroCSV {
	private final static String SEPARADOR = ",";
	private final static String COMILLAS = "\"";

	public static void main(String[] args) {
		BufferedReader lectorCSV = null;

		try {
			lectorCSV = new BufferedReader(new FileReader("./datos.csv"));
			String linea = lectorCSV.readLine();
			while (null != linea) {

				String[] campos = linea.split(SEPARADOR);
				campos = eliminarComillasEnExtremos(campos);
				System.out.println(Arrays.toString(campos));
				linea = lectorCSV.readLine();
			}
		} catch (Exception e) {
			System.out.println("Error al leer: ");
			e.printStackTrace();
		} finally {
			try {
				lectorCSV.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}

	private static String[] eliminarComillasEnExtremos(String[] campos) {
		String result[] = new String[campos.length];
		// ^" es una expresion regular que indica que la cadena empieza por "
		// "$ es una expresion regular que indica que la cadena termina por "
		for (int i = 0; i < result.length; i++) {
			result[i] = campos[i].replaceAll("^" + COMILLAS,
"").replaceAll(COMILLAS + "$", "");
		}
		return result;
	}
```

Caso 2. Lectura de un CSV simple con Scanner

```
private final static String SEPARADOR=",";//se necesita String para split()

private final static char COMILLA='"';

    String dir = System.getProperty("user.dir");
    Scanner lectorCSV;

    try {
        lectorCSV = new Scanner(new File(dir + File.separator +
```

```
"datos.csv")).useDelimiter("
");
        while (lectorCSV.hasNext()){
            String[] campos = lectorCSV.next().split(SEPARADOR);
            campos = eliminarComillasEnExtremos(campos);
            //  System.out.println(Arrays.toString(campos));
        }
    } catch (FileNotFoundException e) {
        // ...
    }
```

Caso 3. Lectura de un CSV complejo con OpenCSV

Como ejemplo de documento CSV complejo, podemos utilizar el siguiente:

```
"Pérez García, ""Torbellino"" Juan","33","Córdoba","1996"
"Gutiérrez Tolosa, ""Ana","21","Granada","1994"
"Rodríguez Antúnez, Pedro","26","Cuenca","1997"
```

La librería OpenCSV permite leer documentos CSV complejos. Esta librería no está incluída en Java, y tiene varias dependencias, que hace que su instalación sea compleja, salvo que dejemos esta tarea a *Maven*. En mi caso, voy a utilizar la versión 4, por ser la que mejor está documentada en http://opencsv.sourceforge.net/.

Para instalar *Maven* podemos seguir las indicaciones en https://mkyong.com/maven/how-to-install-maven-in-windows/.

En el POM, incluimos la dependencia siguiente:

```
    <!-- https://mvnrepository.com/artifact/com.opencsv/opencsv -->
    <dependency>
      <groupId>com.opencsv</groupId>
      <artifactId>opencsv</artifactId>
      <version>4.1</version>
    </dependency>
```

El código sería (aproximadamente) como sigue:

```
    private final static char SEPARADOR=',';
    private final static char COMILLA='"';

        String dir = System.getProperty("user.dir");

       try {
            FileReader lectorArchivo = new FileReader(dir + File.separator
```

```
+ "datos.csv");

            CSVParser parserConPuntoYComa = new
CSVParserBuilder().withSeparator(SEPARADOR)
                    .withQuoteChar(COMILLA)
                    .build();

            CSVReader lectorCSV = new
CSVReaderBuilder(lectorArchivo).withCSVParser(parserConPuntoYComa)
                    .build();

            List<String[]> filas = lectorCSV.readAll();

            for (String[] fila : filas) {
                for (int i=0; i<fila.length; i++)
                    System.out.print("[" + fila[i] + "]");
                System.out.println("");
            }
            //System.out.println(filas);
        } catch (Exception e){
            e.printStackTrace();
        }
```

Escribir un archivo CSV

Para escribir un archivo CSV, se puede utilizar cualquiera de los métodos existentes para escribir archivos (como *FileWriter*) o bien OpenCSV. Me voy a centrar en esta última opción, ya que escribir con *FileWriter* es trivial.

> Los archivos son estructuras secuenciales, y por ello no es posible editar un archivo CSV de manera trivial. Por ello, hay que escribir el archivo completo con las modificaciones que se hayan hecho sobre los datos. Debido a que no es una operación eficiente, cuando se requieren operaciones de escritura, se suelen utilizar bases de datos.

```
    public String csvWriterAll(List<String[]> stringArray, String fichero)
throws Exception {
        CSVWriter writer = new CSVWriter(new FileWriter(fichero));
        writer.writeAll(stringArray);
        writer.close();
        return  Helpers.readFile(path);
    }
```

Para practicar con documento CSV

Contamos con dos documentos CSV, uno simple (llamado simple.csv) y otro complejo (llamado complejo.csv). Crea dichos documentos siguiendo las indicaciones en este documento. Lee dichos documentos y muestra por pantalla su contenido, separando claramente los campos:

```
FILA 1:
    CAMPO 1: [....] CAMPO 2: [....] CAMPO 3: [....] ...
FILA 2:
    ...
```