**CSV Files**

* The CSV stands for Comma-Separated Values.
* It is a simple file format which is used to store tabular data in simple text form, like: spreadsheet or database.
* We can create CSV files in excel or notepad.
* We can export or import this file from M.S office or excel.
* Here the data is stored in column by column, and split by a separator i.e “ , “ (comma).
* Example: [Amar, 42, CSE, pass, 8.6]

**Handling CSV Files in Java**

* First step is to create a CSV File using: 1) Excel 2) Notepad.
* And then select the **save as type** menu CSV (Common delimited) and then click **save** button.
* In notepad you have save the file with the **.csv** extension.

**Reading CSV Files in Java**

* There are three ways to read CSV file:
  + **Scanner** class
  + **String.split()**
  + Using **OpenCSV** API

Java Scanner class:

* It provides different methods by which we can read CSV files in java.
* Scanner class provides a constructor that produces values scanned from specific file. It breaks the input into tokens whenever it finds whitespace.
* Ex: “Hello world” it splits into two tokens: “Hello”,

” world”.

* We can read each token using some next() methods like:
  + next()
  + nextInt()
  + nextDouble()
* It uses a delimiter pattern based on spaces, tabs, newlines. Use useDelimiter() to specify some different delimiter i.e comma for CSV files.
* Example :

package fileHandling;

import java.io.\*;

import java.util.Scanner;

public class UsingScannerClass {

public static void main(String[] args) {

try

{

Scanner object = new Scanner(new File("D:\\CSV file.csv"));

object.useDelimiter(" , ");

while(object.hasNext())

{

System.out.print(object.next());

}

object.close();

}

catch(Exception e)

{

System.out.print("Catch block");

e.getCause();

}

}

}

Split() Method:

* It is used to split a string into an array of substrings or a line into array of string based on some delimiter.
* It uses some delimiter (any character) to split the line.
* Syntax:

String[] result = splitStringObject.split(delimiter);

* Example: {“joe, doe, developer”} this is a line. Here the method identifies each comma in the line and uses it as boundary to create a new element in a array. i.e {“joe” , “doe” , “developer”}
* Example:

package exceloperations;

//BufferedReader and FileReader used in reading files

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

//this class is demonstrating how to read a CSV File

public class BufferReaderClass {

public static void main(String[] args) {

// TODO Auto-generated method stub

//line variable holds each line, read from the csv file

String line="";

//splitBy here is a delimiter used to split each line

String splitBy=",";

//exception block

try

{

//BufferReader reads and provides buffering.FileReader is specified with file path and is used to read the file

BufferedReader bufferobject= new BufferedReader(new FileReader("D:\\CSV file.csv"));

//while loop reads each line from the file with the help of readLine() until the end of the file

while((line=bufferobject.readLine())!=null)

{

//here split() method is used

String[] employee = line.split(splitBy);

if(employee.length==6)

{

System.out.println("Employee [First Name= "+employee[0]+", Last Name= "+employee[1]+", Designation= "+employee[2]+", Contact= "+employee[3]+", Salary= "+employee[4]+", City= "+employee[5]+"]");

}

else

{

System.out.println("Incomplete line");

}

}

}

catch(Exception e)

{

System.out.println("Catch block ");

e.getCause();

e.printStackTrace();

}

}

}

Using OpenCSV API:

* It is third party app.
* OpenCSV is an easy to use CSV(comma-separated values) parser library for java.
* Java language does not provide any native support for effectively handling CSV files, so we are using OpenCSV for handling CSV files in java.
* OpenCSV has some classes like:
  + 1. CSVReader: To read a CSV file.
    2. CSVWriter etc: To Write a CSV file.
* We can READ Csv file in two ways:
  + 1. Read data line by line: by initializing CSVReader object and with the help of readNext() method.
* Example snippet code:

// Java code to illustrate reading a

// CSV file line by line

public static void readDataLineByLine(String file)

{

try {

// Create an object of filereader

// class with CSV file as a parameter.

FileReader filereader = new FileReader(file);

// create csvReader object passing

// file reader as a parameter

CSVReader csvReader = new CSVReader(filereader);

String[] nextRecord;

// we are going to read data line by line

while ((nextRecord = csvReader.readNext()) != null) {

for (String cell : nextRecord) {

System.out.print(cell + "\t");

}

System.out.println();

}

}

catch (Exception e) {

e.printStackTrace();

}

}

* Second method is to Read all data at once: using method called readAll() to read all the records at once into a List.
* While we read csv file by default, the header will not be ignored.
* When we need to skip the first element in the list then we can specify start line while creating CSVReader.
  + 1. CSVReader csvReader = new CSVReaderBuilder(reader).withSkipLines(1)

.build();

* Example snippet code:

// Java code to illustrate reading a

// all data at once

public static void readAllDataAtOnce(String file)

{

try {

// Create an object of file reader

// class with CSV file as a parameter.

FileReader filereader = new FileReader(file);

// create csvReader object and skip first Line

CSVReader csvReader = new CSVReaderBuilder(filereader)

.withSkipLines(1)

.build();

List<String[]> allData = csvReader.readAll();

// print Data

for (String[] row : allData) {

for (String cell : row) {

System.out.print(cell + "\t");

}

System.out.println();

}

}

catch (Exception e) {

e.printStackTrace();

}

}

* WRITING A CSV FILE:
  + 1. Create an instance of CSVWriter by passing FileWriter object as parameter and start writing data to CSV file using methods of CSVWriter Class.
    2. After writing data we need to close CSVWriter connection by calling close() method of CSVWriter class.
* We can Writing Csv file in two ways:
  + 1. Write Data Line by line:
       - Here CSV writer can write line by line by using writeNext() method where a string array is passed with each comma-separated element as a separate entry.
* Example snippet code:

public static void writeDataLineByLine(String filePath)

{

// first create file object for file placed at location

// specified by filepath

File file = new File(filePath);

try {

// create FileWriter object with file as parameter

FileWriter outputfile = new FileWriter(file);

// create CSVWriter object filewriter object as parameter

CSVWriter writer = new CSVWriter(outputfile);

// adding header to csv

String[] header = { "Name", "Class", "Marks" };

writer.writeNext(header);

// add data to csv

String[] data1 = { "Aman", "10", "620" };

writer.writeNext(data1);

String[] data2 = { "Suraj", "10", "630" };

writer.writeNext(data2);

// closing writer connection

writer.close();

}

catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

* Write all Data at once:
  + For writing data at once call writeAll() method of CSVWriter class and pass a List of String[] as the parameter with each String[] representing a line of the file.
  + Example snippet code:

public static void writeDataAtOnce(String filePath)

{

// first create file object for file placed at location

// specified by filepath

File file = new File(filePath);

try {

// create FileWriter object with file as parameter

FileWriter outputfile = new FileWriter(file);

// create CSVWriter object filewriter object as parameter

CSVWriter writer = new CSVWriter(outputfile);

// create a List which contains String array

List<String[]> data = new ArrayList<String[]>();

data.add(new String[] { "Name", "Class", "Marks" });

data.add(new String[] { "Aman", "10", "620" });

data.add(new String[] { "Suraj", "10", "630" });

writer.writeAll(data);

// closing writer connection

writer.close();

}

catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();