import java.io.BufferedWriter;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.text.NumberFormat;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.InputMismatchException;

import java.util.NoSuchElementException;

import java.util.Scanner;

/\*\*

\* @author Samuel Swedberg

\* @version 11/14/22

\*

\* A client that demonstrates reading from a text file and printing ascii tables, and writing files

\*/

public class Client {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String directory = "", answer = "";

boolean tryAgain = true;

while(tryAgain)

{

boolean inValidInput = true;

//Loops until valid input is found

while(inValidInput)

{

System.out.println("Enter starting directory: ");

directory = sc.nextLine();

try {

if(!new File(directory).exists())

throw new FileNotFoundException("You must only enter a directory.");

inValidInput = false;

// FileNotFoundException

} catch (FileNotFoundException fnfe) {

System.out.println(fnfe.getMessage() + " Press a key to continue or N to leave: ");

answer = sc.nextLine();

// Asks user if they want to continue

if(answer.equalsIgnoreCase("N"))

tryAgain = false;

break;

}

}

// Runs only if valid input is found

if(!inValidInput)

{

findFile(directory);

System.out.println("Press a key to continue or N to leave: ");

answer = sc.nextLine();

// Asks user if they want to continue

if(answer.equalsIgnoreCase("N"))

tryAgain = false;

}

}

}

/\*\*

\* @param directory

\*/

public static void findFile(String directory)

{

File file = new File(directory);

File[] files = file.listFiles();

LinkedQueue queueTotal = new LinkedQueue();

LinkedQueue queueTotalSuccess = new LinkedQueue();

LinkedQueue queueTotalError = new LinkedQueue();

// Runs only if files exist

if(files != null)

{

// For loop to run through all files

for(int i=0; i<files.length; i++)

{

File f = files[i];

// Checks if a directory or file

if(f.isDirectory())

findFile(f.getAbsolutePath());

try

{

Scanner scanLine = new Scanner( f );

String fileNameArr = f.getName(); // Exchange Abbreviation

String[] split = fileNameArr.split("\_", 2);

String fileName = split[0];

String absPath = f.getAbsolutePath();

long lastModLong = f.lastModified();

Date lastMod = new Date(lastModLong);

SimpleDateFormat df2 = new SimpleDateFormat("yyyymmdd");

String lastModText = df2.format(lastMod);

int skipFirstLine = 0;

// Scans line

while( scanLine.hasNext() )

{

if(skipFirstLine == 0)

{

skipFirstLine++;

scanLine.next();

}

else

{

LinkedQueue entryQueue = new LinkedQueue();

String stringLine = scanLine.nextLine();

Scanner parse = new Scanner(stringLine);

parse.useDelimiter(",");

try

{

NumberFormat nf = NumberFormat.getInstance();

nf.setMaximumFractionDigits(2);

entryQueue.enqueue(fileName); // Enqueues ex abb

String name, ticker;

name = parse.next();

entryQueue.enqueue(name); // Parses name as string and enqueues

ticker = parse.next();

entryQueue.enqueue(ticker); // Parses ticker as string and enqueues

int date, volume;

date = parse.nextInt();

entryQueue.enqueue(date); // Parses date as int

double open, high, low, close;

open = parse.nextDouble();

entryQueue.enqueue(nf.format(open)); // Parses open as double and enqueues

high = parse.nextDouble();

entryQueue.enqueue(nf.format(high)); // Parses high as double and enqueues

low = parse.nextDouble();

entryQueue.enqueue(nf.format(low)); // Parses low as double and enqueues

close = parse.nextDouble();

entryQueue.enqueue(nf.format(close)); // Parses close as double and enqueues

volume = parse.nextInt();

entryQueue.enqueue(volume); // Parses volume as int and enqueues

entryQueue.enqueue(lastModText);

entryQueue.enqueue(absPath);

queueTotal.enqueue(entryQueue);

String successString = fileName + ", " + name + ", " + ticker + ", " + date + ", " + nf.format(open) + ", " + nf.format(high) + ", " + nf.format(low) + ", " + nf.format(close) + ", " + volume + ", " + lastModText + ", " + absPath;

queueTotalSuccess.enqueue(successString);

if(parse.hasNext())

{

System.out.println("Too many records");

}

}

catch ( InputMismatchException ime )

{

System.out.println("Error in data format");

queueTotalError.enqueue(stringLine);

}

catch ( NoSuchElementException nse )

{

System.out.println("No more tokens");

queueTotalError.enqueue(stringLine);

}

catch ( IllegalStateException ise )

{

ise.printStackTrace();

}

parse.close();

}

}

scanLine.close();

}

catch ( IOException ioe )

{

ioe.printStackTrace();

}

}

}

printTable(queueTotal);

writeFile(queueTotalSuccess, "A:\\EODdata.txt");

writeFile(queueTotalError, "A:\\EODdataERROR.txt");

}

/\*\*

\* @param queueTotal

\*/

public static void printTable(LinkedQueue queueTotal)

{

ASCII table = new ASCII();

table.getColumns().add(new ASCII.Column("Exchange"));

table.getColumns().add(new ASCII.Column("Name"));

table.getColumns().add(new ASCII.Column("Ticker"));

table.getColumns().add(new ASCII.Column("Date"));

table.getColumns().add(new ASCII.Column("Open"));

table.getColumns().add(new ASCII.Column("High"));

table.getColumns().add(new ASCII.Column("Low"));

table.getColumns().add(new ASCII.Column("Close"));

table.getColumns().add(new ASCII.Column("Volume"));

table.getColumns().add(new ASCII.Column("Last Modified"));

table.getColumns().add(new ASCII.Column("Absolute Path"));

String exAb, name, ticker, lastMod, absPath, open, high, low, close, date, volume;

int trueSize = queueTotal.size();

for(int i=0; i<trueSize; i++)

{

LinkedQueue queue = ( LinkedQueue ) queueTotal.dequeue();

ASCII.Row row = new ASCII.Row();

/\*

System.out.println("exAb: " + queue.dequeue());

System.out.println("name: " + queue.dequeue());

System.out.println("ticker: " + queue.dequeue());

System.out.println("date: " + queue.dequeue());

System.out.println("open: " + queue.dequeue());

System.out.println("high: " + queue.dequeue());

System.out.println("low: " + queue.dequeue());

System.out.println("close: " + queue.dequeue());

System.out.println("volume: " + queue.dequeue());

System.out.println("lastMod: " + queue.dequeue());

System.out.println("absPath: " + queue.dequeue());\*/

exAb = ( String ) queue.dequeue();

name = ( String ) queue.dequeue();

ticker = ( String ) queue.dequeue();

int intDate = ( int ) queue.dequeue();

date = Integer.toString(intDate);

open = ( String ) queue.dequeue();

high = ( String ) queue.dequeue();

low = ( String ) queue.dequeue();

close = ( String ) queue.dequeue();

int intVolume = ( int ) queue.dequeue();

volume = Integer.toString(intVolume);

lastMod = ( String ) queue.dequeue();

absPath = ( String ) queue.dequeue();

table.getData().add(row);

row.getValues().add(exAb);

row.getValues().add(name);

row.getValues().add(ticker);

row.getValues().add(date);

row.getValues().add(open);

row.getValues().add(high);

row.getValues().add(low);

row.getValues().add(close);

row.getValues().add(volume);

row.getValues().add(lastMod);

row.getValues().add(absPath);

}

table.calcWidth();

table.printTable();

}

/\*\*

\* @param queue, file

\*/

public static void writeFile(LinkedQueue queue, String file)

{

int trueSize = queue.size();

try

{

FileWriter fileWriter = new FileWriter(file);

BufferedWriter writer = new BufferedWriter(fileWriter);

for(int i=0;i<trueSize;i++)

{

String s = ( String ) queue.dequeue();

writer.write(s);

writer.newLine();

}

writer.close();

}

catch( IOException ioe )

{

ioe.printStackTrace();

}

}

}

import java.util.ArrayList;

import java.util.List;

/\*\*

\* @author Samuel Swedberg

\* @version 11/14/22

\*

\* The ASCII class holds information related to printing ASCII Tables

\*/

public class ASCII {

private final List<Column> columns = new ArrayList<>();

private final List<Row> data = new ArrayList<>();

private int maxWidth = Integer.MAX\_VALUE;

/\*\*

\* @return columns

\*/

public List<Column> getColumns() {

return columns;

}

/\*\*

\* @return data

\*/

public List<Row> getData() {

return data;

}

/\*\*

\* @return maxWidth

\*/

public int getNaxWidth() {

return maxWidth;

}

/\*\*

\* @param maxWidth

\*/

public void setMaxWidth(int maxWidth) {

this.maxWidth = maxWidth;

}

public static class Column {

private String name;

private int width;

public Column(String name) {

this.name = name;

}

}

public static class Row {

private final List<String> values = new ArrayList<>();

public List<String> getValues() {

return values;

}

}

/\*\*

\* Calculates width

\*/

public void calcWidth() {

for (Column column : columns) {

column.width = column.name.length() + 1;

}

for (Row row : data) {

int colIdx = 0;

for (String value : row.values) {

Column column = columns.get(colIdx);

if (value == null) continue;

column.width = Math.max(column.width, value.length() + 1);

colIdx++;

}

}

for (Column column : columns) {

column.width = Math.min(column.width, maxWidth);

}

}

/\*\*

\* @param columns, sb

\*/

private void writeColumnNames(List<Column> columns, StringBuilder sb) {

sb.append("| ");

for (Column column : columns) {

sb.append(String.format(" %-" + (column.width) + "s", column.name));

sb.append(" |");

}

sb.append("\n");

}

/\*\*

\* @param columns, sb

\*/

private void writeSeparator(List<Column> columns, StringBuilder sb) {

sb.append("+-");

for (Column column : columns) {

sb.append(String.format(" %-" + (column.width + 1) + "s", "").replace(' ', '-'));

sb.append("-+");

}

sb.append("\n");

}

/\*\*

\* @param columns, rows, sb

\*/

private void writeValues(List<Column> columns, List<Row> rows, StringBuilder sb) {

for (Row row : rows) {

int columnIndex = 0;

sb.append("| ");

for (String value : row.values) {

if (value != null && value.length() > maxWidth)

value = value.substring(0, maxWidth - 1);

sb.append(String.format(" %-" + columns.get(columnIndex).width + "s", value));

sb.append(" |");

columnIndex++;

}

sb.append("\n");

}

}

/\*\*

\* prints table

\*/

public void printTable() {

StringBuilder sb = new StringBuilder();

writeSeparator(columns, sb);

writeColumnNames(columns, sb);

writeSeparator(columns, sb);

writeValues(columns, data, sb);

writeSeparator(columns, sb);

System.out.println(sb.toString());

}

}

**+-------------+----------------------------------------------------+------------+------------+-------------+-------------+-------------+-------------+--------------+-----------------+-------------------------------------------+**

**| Exchange | Name | Ticker | Date | Open | High | Low | Close | Volume | Last Modified | Absolute Path |**

**+-------------+----------------------------------------------------+------------+------------+-------------+-------------+-------------+-------------+--------------+-----------------+-------------------------------------------+**

**| AMEX | First Priority Clo Bond ETF | AAA | 20221005 | 24.1 | 24.1 | 24.1 | 24.1 | 100 | 20222314 | A:\EODdata\20221005\AMEX\_20221005.txt |**

**| AMEX | GS Physical Gold ETF | AAAU | 20221005 | 16.98 | 17.06 | 16.89 | 17.03 | 188600 | 20222314 | A:\EODdata\20221005\AMEX\_20221005.txt |**

**| AMEX | Almaden Minerals | AAU | 20221005 | 0.24 | 0.24 | 0.23 | 0.24 | 185000 | 20222314 | A:\EODdata\20221005\AMEX\_20221005.txt |**

**| AMEX | Absolute Core Strategy ETF | ABEQ | 20221005 | 25.89 | 26.11 | 25.86 | 26.06 | 13800 | 20222314 | A:\EODdata\20221005\AMEX\_20221005.txt |**

**| AMEX | Alps Clean Energy ETF | ACES | 20221005 | 58.18 | 58.18 | 55.6 | 56.72 | 54700 | 20222314 | A:\EODdata\20221005\AMEX\_20221005.txt |**

**| AMEX | Aptus Collared Income Opportunity ETF | ACIO | 20221005 | 28.83 | 29.05 | 28.8 | 28.97 | 84900 | 20222314 | A:\EODdata\20221005\AMEX\_20221005.txt |**

**| AMEX | American Customer Satisfaction Core Alpha ETF | ACSI | 20221005 | 43.12 | 43.12 | 43.12 | 43.12 | 100 | 20222314 | A:\EODdata\20221005\AMEX\_20221005.txt |**

AMEX, First Priority Clo Bond ETF, AAA, 20221005, 24.1, 24.1, 24.1, 24.1, 100, 20222314, A:\EODdata\20221005\AMEX\_20221005.txt

AMEX, GS Physical Gold ETF, AAAU, 20221005, 16.98, 17.06, 16.89, 17.03, 188600, 20222314, A:\EODdata\20221005\AMEX\_20221005.txt

AMEX, Almaden Minerals, AAU, 20221005, 0.24, 0.24, 0.23, 0.24, 185000, 20222314, A:\EODdata\20221005\AMEX\_20221005.txt

AMEX, Absolute Core Strategy ETF, ABEQ, 20221005, 25.89, 26.11, 25.86, 26.06, 13800, 20222314, A:\EODdata\20221005\AMEX\_20221005.txt

AMEX, Alps Clean Energy ETF, ACES, 20221005, 58.18, 58.18, 55.6, 56.72, 54700, 20222314, A:\EODdata\20221005\AMEX\_20221005.txt

AMEX, Aptus Collared Income Opportunity ETF, ACIO, 20221005, 28.83, 29.05, 28.8, 28.97, 84900, 20222314, A:\EODdata\20221005\AMEX\_20221005.txt

AMEX, American Customer Satisfaction Core Alpha ETF, ACSI, 20221005, 43.12, 43.12, 43.12, 43.12, 100, 20222314, A:\EODdata\20221005\AMEX\_20221005.txt

AMEX, Leadershares Activist Leaders ETF, ACTV, 20221005, 28.15, 28.4, 27.89, 28.4, 5000, 20222314, A:\EODdata\20221005\AMEX\_20221005.txt

AMEX, Acme United Corp, ACU, 20221005, 23.76, 24.24, 23.05, 23.73, 14800, 20222314, A:\EODdata\20221005\AMEX\_20221005.txt

AMEX, American Conservative Values ETF, ACVF, 20221005, 28.25, 28.54, 28.04, 28.39, 7100, 20222314, A:\EODdata\20221005\AMEX\_20221005.txt

Total Advancing Stocks,ADVT.IDX,20221005,981,2265,761,1893,3113654000

NYSE Declining Stocks,DECN.IDX,20221005,1675,1805,1196,1354,2500065000

Total Declining Stocks,DECT.IDX,20221005,4007,4413,3108,3472,4475918000

S&P 500,SP500.IDX,20221005,3753.25,3806.91,3722.66,3783.28,2502482000

OTC US Unchanged Stocks,UNCU.IDX,20221005,226,449,226,449,4513400000

Richmond Mutual Bancorp Inc,RMBI,20221005,13.7001,13.71,13.7001,13.71

Rumble Inc WT,RUMBW,3.87,3.87,3.32,3.55,74300,88