import java.util.ArrayList;

import java.util.Arrays;

import java.net.URL;

import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileWriter;

import java.io.Reader;

import java.io.StringReader;

import org.apache.commons.csv.CSVFormat;

import org.apache.commons.csv.CSVParser;

/\*\*

\* The <code>FileResource</code> class represents a file and allows access to its contents a line at

\* a time, using the method <code>lines</code>, or a word at a time, using the method

\* <code>words</code>. These strings can then be iterated over using a <code>for</code> loop.

\*

\* <P>

\* Example usage:

\*

\* <PRE>

\* FileResource fr = new FileResource();

\* for (String s : fr.words()) {

\* // print or process s

\* }

\* </PRE>

\*

\* <P>

\* If each line of the file represents separated data values, because its a CSV file, then the user

\* can get a <code>getCSVParser</code> object to access that data more directly, using one of the

\* <code>getCSVParser</code> methods.

\*

\* <P>

\* Example CSV usage:

\*

\* <PRE>

\* FileResource fr = new FileResource("food.csv");

\* for (CSVRecord record : fr.getCSVParser()) {

\* // print or process fields in record

\* String name = record.get("Name");

\* // other processing

\* }

\* </PRE>

\*

\* <P>

\* This software is licensed with an Apache 2 license, see

\* http://www.apache.org/licenses/LICENSE-2.0 for details.

\*

\* @author Duke Software Team

\*/

public class FileResource {

private String myPath;

private String mySource;

private File mySaveFile;

/\*\*

\* Create a <code>FileResource</code> object that opens the file chosen by the user using a file

\* selection dialog box.

\*

\* @throws exception if no file is selected by the user

\*/

public FileResource () {

initRead();

}

/\*\*

\* Create a <code>FileResource</code> object that opens a file represented by the File object

\* passed as a parameter.

\*

\* Useful, for example, when used in conjunction with the <code>DirectoryResource</code> class.

\*

\* @param file the file to be represented by this resource

\* @throws exception if the file cannot be accessed

\*/

public FileResource (File file) {

initRead(file);

}

/\*\*

\* Create a <code>FileResource</code> object that opens a file whose name is passed as a

\* parameter.

\*

\* The named file should be on the current class path to be found.

\*

\* @param filename the name of the file to be opened

\* @throws exception if the filename cannot be accessed

\*/

public FileResource (String filename) {

initRead(filename);

}

/\*\*

\* Create a <code>FileResource</code> object that opens the file chosen by the user using a file

\* selection dialog box, possibly to write to it.

\*

\* If the user wants to change the contents of the open file by using the method

\* <code>write</code> to add new strings to it, pass <code>true</code> as the second parameter.

\* Otherwise it is assumed the user will only iterate over the existing contents of the file.

\*

\* @param writable allow changes to this file only if true

\* @throws exception if no file is selected by the user

\*/

public FileResource (boolean writable) {

if (writable) {

initWrite();

}

else {

initRead();

}

}

/\*\*

\* Create a <code>FileResource</code> object that opens a file represented by the File object

\* passed as a parameter, possibly to write to it.

\*

\* If the user wants to change the contents of the open file by using the method

\* <code>write</code> to add new strings to it, pass <code>true</code> as the second parameter.

\* Otherwise it is assumed the user will only iterate over the existing contents of the file.

\*

\* Useful, for example, when used in conjunction with the <code>DirectoryResource</code> class.

\*

\* @param file the file to be represented by this resource

\* @param writable allow changes to this file only if true

\* @throws exception if the file cannot be accessed

\*/

public FileResource (File file, boolean writable) {

if (writable) {

initWrite(file);

}

else {

initRead(file);

}

}

/\*\*

\* Create a <code>FileResource</code> object that opens a file whose name is passed as a

\* parameter, possibly to write to it.

\*

\* If the user wants to change the contents of the open file by using the method

\* <code>write</code> to add new strings to it, pass <code>true</code> as the second parameter.

\* Otherwise it is assumed the user will only iterate over the existing contents of the file.

\*

\* The named file should be on the current class path to be found.

\*

\* @param filename the name of the file to be opened

\* @param writable allow changes to this file only if true

\* @throws exception if the filename cannot be accessed

\*/

public FileResource (String filename, boolean writable) {

if (writable) {

initWrite(filename);

}

else {

initRead(filename);

}

}

/\*\*

\* Allow access to this opened file one line at a time.

\*

\* @return an <code>Iterable</code> that will allow access to contents of opened file one line

\* at a time.

\*/

public Iterable<String> lines () {

return new TextIterable(mySource, "\\n");

}

/\*\*

\* Allow access to this opened file one word at a time, where words are separated by

\* white-space. This means any form of spaces, like tabs or newlines, can delimit words.

\*

\* @return an <code>Iterable</code> that will allow access to contents of opened file one word

\* at a time.

\*/

public Iterable<String> words () {

return new TextIterable(mySource, "\\s+");

}

/\*\*

\* Return entire contents of this opened file as one string.

\*

\* @return a <code>String</code> that is the contents of the open file

\*/

public String asString () {

return mySource;

}

/\*\*

\* Returns a <code>CSVParser</code> object to access the contents of an open file.

\*

\* Each line of the file should be formatted as data separated by commas and with a header row

\* to describe the column names.

\*

\* @return a <code>CSVParser</code> that can provide access to the records in the file one at a

\* time

\* @throws exception if this file does not represent a CSV formatted data

\*/

public CSVParser getCSVParser () {

return getCSVParser(true);

}

/\*\*

\* Returns a <code>CSVParser</code> object to access the contents of an open file, possibly

\* without a header row.

\*

\* Each line of the file should be formatted as data separated by commas and with/without a

\* header row to describe the column names.

\*

\* @param withHeader uses first row of data as a header row only if true

\* @return a <code>CSVParser</code> that can provide access to the records in the file one at a

\* time

\* @throws exception if this file does not represent a CSV formatted data

\*/

public CSVParser getCSVParser (boolean withHeader) {

return getCSVParser(withHeader, ",");

}

/\*\*

\* Returns a <code>CSVParser</code> object to access the contents of an open file, possibly

\* without a header row and a different data delimiter than a comma.

\*

\* Each line of the file should be formatted as data separated by the delimiter passed as a

\* parameter and with/without a header row to describe the column names. This is useful if the

\* data is separated by some character other than a comma.

\*

\* @param withHeader uses first row of data as a header row only if true

\* @param delimiter a single character that separates one field of data from another

\* @return a <code>CSVParser</code> that can provide access to the records in the file one at a

\* time

\* @throws exception if this file does not represent a CSV formatted data

\* @throws exception if <code>delimiter.length() != 1</code>

\*/

public CSVParser getCSVParser (boolean withHeader, String delimiter) {

if (delimiter == null || delimiter.length() != 1) {

throw new ResourceException("FileResource: CSV delimiter must be a single character: " + delimiter);

}

try {

char delim = delimiter.charAt(0);

Reader input = new StringReader(mySource);

if (withHeader) {

return new CSVParser(input, CSVFormat.EXCEL.withHeader().withDelimiter(delim));

}

else {

return new CSVParser(input, CSVFormat.EXCEL.withDelimiter(delim));

}

}

catch (Exception e) {

throw new ResourceException("FileResource: cannot read " + myPath + " as a CSV file.");

}

}

/\*\*

\* Allows access to the column names of the header row of a CSV file (the first line in the

\* file) one at a time. If the CSV file did not have a header row, then an empty

\* <code>Iterator</code> is returned.

\*

\* @param parser the <code>CSVParser</code> that has been created for this file

\* @return an <code>Iterable</code> that allows access one header name at a time

\*/

public Iterable<String> getCSVHeaders (CSVParser parser) {

return parser.getHeaderMap().keySet();

}

/\*\*

\* Writes a string to the end of this file.

\*

\* @param s the string to saved to the file

\*/

public void write (String s) {

ArrayList<String> list = new ArrayList<String>();

list.add(s);

write(list);

}

/\*\*

\* Writes a list of strings to the end of this file, one element per line.

\*

\* @param list the strings to saved to the file

\*/

public void write (StorageResource list) {

// we know it is an ArrayList underneath

write((ArrayList<String>)(list.data()));

}

/\*\*

\* Writes a list of strings to the end of this file, one element per line.

\*

\* @param list the strings to saved to the file

\*/

public void write (String[] list) {

// BUGBUG: yuck :(

write(new ArrayList<String>(Arrays.asList(list)));

}

/\*\*

\* Writes a list of strings to the end of this file, one element per line.

\*

\* @param list the strings to saved to the file

\*/

public void write (ArrayList<String> list) {

if (mySaveFile != null) {

// build string to save

StringBuilder sb = new StringBuilder();

for (String s : list) {

sb.append(s);

sb.append("\n");

}

// save it locally (so it can be read later)

mySource += sb.toString();

// save it externally to the file

PrintWriter writer = null;

try {

writer = new PrintWriter(new FileWriter(mySaveFile, true));

writer.println(sb.toString());

}

catch (Exception e) {

throw new ResourceException("FileResource: cannot change " + mySaveFile);

}

finally {

try {

if (writer != null) {

writer.close();

}

}

catch (Exception e) {

// should never happen

}

}

}

}

// Prompt user for file to open

private void initRead () {

File f = FileSelector.selectFile();

if (f == null) {

throw new ResourceException("FileResource: no file choosen for reading");

}

else {

initRead(f);

}

}

// Create from a given File

private void initRead (File f) {

try {

initRead(f.getCanonicalPath());

}

catch (Exception e) {

throw new ResourceException("FileResource: cannot access " + f);

}

}

// Create from the name of a File

private void initRead (String fname) {

try {

myPath = fname;

InputStream is = getClass().getClassLoader().getResourceAsStream(fname);

if (is == null) {

is = new FileInputStream(fname);

}

mySource = initFromStream(is);

}

catch (Exception e) {

throw new ResourceException("FileResource: cannot access " + fname);

}

}

// store data (keep in sync with URLResource)

private String initFromStream (InputStream stream) {

BufferedReader buff = null;

try {

buff = new BufferedReader(new InputStreamReader(stream, "UTF-8"));

StringBuilder contents = new StringBuilder();

String line = null;

while ((line = buff.readLine()) != null) {

contents.append(line + "\n");

}

return contents.toString();

}

catch (Exception e) {

throw new ResourceException("FileResource: error encountered reading " + myPath, e);

}

finally {

try {

if (buff != null) {

buff.close();

}

}

catch (Exception e) {

// should never happen

}

}

}

// prompt user for file for writing

private void initWrite () {

File f = FileSelector.saveFile();

if (f == null) {

throw new ResourceException("FileResource: no file choosen for writing");

}

else {

initWrite(f);

}

}

// create file for writing

private void initWrite (File f) {

try {

mySaveFile = f;

if (f.exists() && f.canWrite()) {

initRead(f);

}

else {

mySource = "";

myPath = f.getCanonicalPath();

}

}

catch (Exception e) {

throw new ResourceException("FileResource: cannot access " + f, e);

}

}

// create file for writing

private void initWrite (String fname) {

try {

URL loc = getClass().getClassLoader().getResource(fname);

if (loc != null) {

fname = loc.toString();

}

initWrite(new File(fname));

}

catch (Exception e) {

throw new ResourceException("FileResource: cannot access " + fname);

}

}

}