**public class WordPlay {**

**public static boolean isVowel(char ch){**

**if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||**

**ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' ){**

**return true;**

**}**

**else {**

**return false;**

**}**

**}**

**// tester method**

**public static String replaceVowel(String phrase, char ch){**

**StringBuilder input = new StringBuilder(phrase);**

**// String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";**

**String s = null;**

**for (int i = 0; i < input.length(); i++){**

**char to = '\*';**

**char currch = input.charAt(i);**

**if (currch == 'a' || currch == 'e' || currch == 'i' || currch == 'o' || currch == 'u' || currch == 'A' || currch == 'E' || currch == 'I' || currch == 'O' || currch == 'U' ){ input.setCharAt(i, to);**

**}**

**// otherwise: do nothing**

**}**

**return input.toString();**

**}**

**public static void main(String[] args) {**

**// TODO Auto-generated method stub**

**WordPlay wp = new WordPlay();**

**boolean result = isVowel('F');**

**String resultReplace = replaceVowel("Hello World", '\*' );**

**System.out.println(result);**

**System.out.println(resultReplace);**

**}**

**}**

**public class CommonWords {**

**public String[] getCommon(){**

**FileResource resource = new FileResource("C:\\Users\\Karen Goh Seow Hui\\Documents\\Coursera-Java\\CommonWords\\data\\common.txt");**

**String[] common = new String[20];**

**int index = 0;**

**for(String s : resource.words()){**

**common[index] = s;**

**index += 1;**

**}**

**return common;**

**}**

**public int indexOf(String[] list, String word) {**

**for (int k=0; k<list.length; k++) {**

**if (list[k].equals(word)) {**

**return k;**

**}**

**}**

**return -1;**

**}**

**public void countWords(FileResource resource, String[] common, int[] counts){**

**for(String word : resource.words()){**

**word = word.toLowerCase();**

**int index = indexOf(common,word);**

**if (index != -1) {**

**counts[index] += 1;**

**}**

**}**

**}**

**void countShakespeare(){**

**//String[] plays = {"caesar.txt", "errors.txt", "hamlet.txt",**

**// "likeit.txt", "macbeth.txt", "romeo.txt"};**

**// String[] plays = {"romeo.txt"};**

**String[] common = getCommon();**

**int[] counts = new int[common.length];**

**// for(int k=0; k < plays.length; k++){**

**FileResource resource = new FileResource("C:\\Users\\Karen Goh Seow Hui\\Documents\\Coursera-Java\\CommonWords\\data\\romeo.txt");**

**countWords(resource,common,counts);**

**// System.out.println("done with " + plays[k]);**

**//}**

**for(int k=0; k < common.length; k++){**

**System.out.println(common[k] + "\t" + counts[k]);**

**}**

**}**

**public static void main(String[] args) {**

**// TODO Auto-generated method stub**

**CommonWords cw = new CommonWords();**

**// FileResource newFile = new FileResource("C:\\Users\\Karen Goh Seow Hui\\Documents\\Java\\CommonWords\\romeo.txt");**

**cw.countShakespeare();**

**}**

**}**

**import java.util.\*;**

**import edu.duke.\*;**

**public class WordsWithArrays {**

**StorageResource myWords;**

**public WordsWithArrays() {**

**myWords = new StorageResource();**

**}**

**public void readWords(){**

**myWords.clear();**

**FileResource resource = new FileResource();**

**for(String word : resource.words()){**

**myWords.add(word.toLowerCase());**

**}**

**}**

**public boolean contains(String[] list, String word, int numStored){**

**for(int k=0; k < numStored; k++){**

**if (list[k].equals(word)){**

**return true;**

**}**

**}**

**return false;**

**}**

**public int countDifferentArray(){**

**int numStored = 0;**

**String[] words = new String[myWords.size()];**

**for(String s : myWords.data()){**

**if (! contains(words,s, numStored)){**

**words[numStored] = s;**

**numStored++;**

**}**

**}**

**return numStored;**

**}**

**public void tester(){**

**readWords();**

**System.out.println("number of words read: "+myWords.size());**

**int unique = countDifferentArray();**

**System.out.println("array count "+unique);**

**}**

**}**

**import** java.util.ArrayList;

**public** **class** WordFrequencies {

ArrayList<String>myWords;

ArrayList<Integer>myFreqs;

**public** WordFrequencies(){

myWords = **new** ArrayList<String>();

myFreqs = **new** ArrayList<Integer>();

}

**public** **void** findUnique(){

myWords.clear();

myFreqs.clear();

// read the file

FileResource resource = **new** FileResource("C:\\Users\\Karen Goh Seow Hui\\Documents\\Coursera-Java\\WordFrequencies\\small.txt");

**for** (String s : resource.words()){ // words() -> returns an Iterable that provides access to the contents of this opened file one word @ a time

**int** index = myWords.indexOf(s);

**if** (index == -1){ // if the words are not in the list

myWords.add(s);

myFreqs.add(1);

}**else**{

**if** (index != -1){

**int** Freq = myFreqs.get(index);

myFreqs.set(index, Freq + 1);

// for (int k = 0; k < myWords.size(); k++){

// System.out.println( myWords.get(k));

//System.out.println(myFreqs);

}

}

}

}

**public** **void** tester(){

findUnique();

System.***out***.println("# of unique words " + myWords.size());

**for** (**int** k = 0; k < myWords.size(); k++){

System.***out***.println(myFreqs.get(k) + "\t" + myWords.get(k));

}

**int** index = findMax();

System.***out***.println("The words that occurs most often and its counts are : " + myWords.get(index) + " " + myFreqs.get(index));

}

**public** **int** findMax(){

**int** max = myFreqs.get(0);

**int** maxIndex = 0;

**for**(**int** k = 0; k < myFreqs.size(); k++){

**if**(myFreqs.get(k) > max){

max = myFreqs.get(k);

maxIndex = k;

}

}

**return** maxIndex;

}

**public** **static** **void** main(String[] args) {

WordFrequencies wf = **new** WordFrequencies();

wf.tester();

}

}

Output:

# of unique words 8

1 lots

2 of

1 words

1 are

1 here

3 the

1 a

2 and

The words that occurs most often and its counts are : the 3

**package** edu.duke;

**public** **class** CaeserCipher {

**public** **static** String encrypt(String input, **int** key) {

//Make a StringBuilder with message (encrypted)

StringBuilder encrypted = **new** StringBuilder(input);

//Write down the alphabet

String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

String alphabet1 = "abcdefghijklmnopqrstuvwxyz";

//Compute the shifted alphabet

String shiftedAlphabet = alphabet.substring(key)+

alphabet.substring(0,key);

String shiftedAlphabet1 = alphabet1.substring(key)+ alphabet1.substring(0,key);

//Count from 0 to < length of encrypted, (call it i)

**for**(**int** i = 0; i < encrypted.length(); i++) {

//Look at the ith character of encrypted (call it currChar)

**char** currChar = encrypted.charAt(i);

//Find the index of currChar in the alphabet (call it idx)

**int** idx;

**if**(Character.*isLowerCase*(currChar)){

idx = alphabet1.indexOf(currChar);

}

//if(Character.isUpperCase(currChar)){

**else**{

// idx = alphabet.indexOf(Character.toUpperCase(currChar));

idx = alphabet.indexOf(currChar);

}

**if**(idx != -1){

**if**(Character.*isLowerCase*(currChar)){

**char** newCharUpper = shiftedAlphabet1.charAt(idx);

//Get the idxth character of shiftedAlphabet (newChar)

encrypted.setCharAt(i, newCharUpper);

}**else**{

// if(Character.isUpperCase(currChar)){

//idx = alphabet.indexOf(Character.toUpperCase(currChar));

**char** newChar = shiftedAlphabet.charAt(idx);

encrypted.setCharAt(i, newChar);

}

//Otherwise: do nothing

//Your answer is the String inside of encrypted

}

}

**return** encrypted.toString();

}

**public** **int**[] countLetters(String encrypted){

String alph = "abcdefghijklmnopqrstuvwxyz";

**int**[] counts = **new** **int**[26];

**for** (**int** k = 0; k < encrypted.length(); k++){

**char** ch = Character.*toLowerCase*(encrypted.charAt(k));

**int** dex = alph.indexOf(ch);

**if** (dex != -1){

counts[dex] += 1;

//return counts;

}

}

**return** counts;

}

**public** String decrypt(String encrypted){

//CaeserCipher cc = new CaeserCipher();

**int**[] freqs = countLetters(encrypted);

**int** maxDex = maxIndex(freqs);

**int** dkey = maxDex - 4;

**if** (maxDex < 4){

dkey = 26 - (4- maxDex);

}

**return** *encrypt*(encrypted, 26-dkey);

}

**private** **int** maxIndex(**int**[] freqs) {

**int** maxDex = 0;

**for** (**int** k = 0; k < freqs.length; k++){

**if** (freqs[k] > freqs[maxDex]){

maxDex = k;

}

}

**return** maxDex;

}

**public** **static** **void** testCaesar(){

//public void main (String[]args){

//CaesarCipher cs = new CaesarCipher();

**int** key = 17;

FileResource newFile = **new** FileResource("C:\\Users\\Karen Goh Seow Hui\\Documents\\Java\\CaesarCipher\\message2.txt");

String message = newFile.asString();

// String message = "FIRST LEGION ATTACK EAST FLANK!";

// String message = words.asString();

String encrypted = *encrypt*(message, key);

System.***out***.println(encrypted);

String decrypt = *encrypt*(encrypted, 26-key);

System.***out***.println(decrypt);

}

**public** **static** **void** main (String[]args){

// CaesarCipher cs = new CaesarCipher();

*testCaesar*();

}

}

**Output:**

Uvri Fnve,

Ef drkkvi nyrk pfl drp yrmv yvriu, kyviv zj ef trbv

ze kyv tfewvivetv iffd. Kyv trbv zj r czv. Gcvrjv bvvg

nfibzex fe Tflijvir mzuvfj.

Kyrebj,

Uivn

Dear Owen,

No matter what you may have heard, there is no cake

in the conference room. The cake is a lie. Please keep

working on Coursera videos.

Thanks,

Drew

**import** java.util.ArrayList;

**public** **class** CharactersInPlay {

**private** ArrayList<String> name;

**private** ArrayList<Integer> count;

**public** CharactersInPlay() {

name = **new** ArrayList<String>();

count = **new** ArrayList<Integer>();

}

**public** **void** update(String person){

//if (!name.contains(person))

**int** index = name.indexOf(person);

**if** (index == -1){

name.add(person);

count.add(1);

// System.out.println(name);

} **else** {

**int** freq = count.get(index);

count.set(index,freq+1);

}

}

**public** **void** findAllCharacters(){

name.clear();

count.clear();

FileResource resource = **new** FileResource("C:\\Users\\Karen Goh Seow Hui\\ELunar\\Assignment4\\src\\"

+ "macbethSmall.txt");

**for**(String s : resource.lines()){

**int** period = s.indexOf(".");

**int** index = name.indexOf(s);

// while (s.contains(".")){

// String lineWithName = s.substring(0, period);

**if** (s.contains(".") && (index == -1)){

// if (index == -1 ){

String lineWithName = s.substring(0, period);

update(lineWithName);

// System.out.printl(lineWithName);

// System.out.println(name.size());

index++;

//}

}

}}

**public** **int** findMax(){

**int** max = count.get(0);

**int** maxIndex = 3;

**for**(**int** k=0; k < count.size(); k++){

**if** (count.get(k) > max){

max = count.get(k);

maxIndex = k;

}

}

**return** maxIndex;

}

**public** **void** tester(){

findAllCharacters();

System.***out***.println("# of lines " + name.size());

**for** (**int** k = 0; k < name.size(); k++){

System.***out***.println(name.get(k) + "\t" + count.get(k));

}

// for (int k = 0; k < name.size(); k++){

**int** index = findMax();

//System.out.println(name.get(index));

**for** (**int** k = 0; k < name.size(); k++){

**if** ((count.get(k) == index || count.get(k) >= findMax())){

System.***out***.println(name.get(k)+" "+ count.get(k));

}

}

}

**public** **static** **void** main(String[] args) {

CharactersInPlay cp = **new** CharactersInPlay();

cp.tester();

}

}

Output :

# of lines 6

MACBETH 3

Duncan comes here tonight 1

LADY MACBETH 2

May read strange matters 1

But be the serpent under it 1

Give solely sovereign sway and masterdom 1

MACBETH 3

LADY MACBETH 2

import edu.duke.\*;

import java.util.\*;

public class GladLib {

private ArrayList<String> adjectiveList;

private ArrayList<String> nounList;

private ArrayList<String> colorList;

private ArrayList<String> countryList;

private ArrayList<String> nameList;

private ArrayList<String> animalList;

private ArrayList<String> timeList;

private ArrayList<String> verbList;

private ArrayList<String> fruitList;

private ArrayList<String> used;

private ArrayList<Integer> myFreqs;

private Random myRandom;

private static String dataSourceURL = "http://dukelearntoprogram.com/course3/data";

private static String dataSourceDirectory = "data";

public GladLib(){

initializeFromSource(dataSourceDirectory);

used = new ArrayList<String>();

myFreqs = new ArrayList<Integer>();

myRandom = new Random();

}

public GladLib(String source){

initializeFromSource(source);

used = new ArrayList<String>();

myFreqs = new ArrayList<Integer>();

myRandom = new Random();

}

private void initializeFromSource(String source) {

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

adjectiveList= readIt(source+"/adjective.txt");

nounList = readIt(source+"/noun.txt");

verbList = readIt(source+"/verb.txt");

fruitList = readIt(source +"/fruit.txt");

colorList = readIt(source+"/color.txt");

countryList = readIt(source+"/country.txt");

nameList = readIt(source+"/name.txt");

animalList = readIt(source+"/animal.txt");

timeList = readIt(source+"/timeframe.txt");

}

private String randomFrom(ArrayList<String> source){

int index = myRandom.nextInt(source.size());

return source.get(index);

}

private String getSubstitute(String label) {

if (label.equals("country")) {

return randomFrom(countryList);

}

if (label.equals("color")){

return randomFrom(colorList);

}

if (label.equals("noun")){

return randomFrom(nounList);

}

if (label.equals("name")){

return randomFrom(nameList);

}

if (label.equals("adjective")){

return randomFrom(adjectiveList);

}

if (label.equals("verb")){

return randomFrom(verbList);

}

if (label.equals("fruit")){

return randomFrom(fruitList);

}

if (label.equals("animal")){

return randomFrom(animalList);

}

if (label.equals("timeframe")){

return randomFrom(timeList);

}

if (label.equals("number")){

return ""+myRandom.nextInt(50)+5;

}

return "\*\*UNKNOWN\*\*";

}

private String processWord(String w){

int first = w.indexOf("<");

int last = w.indexOf(">", first);

if(first == 1 || last == -1){

return w;

}

String prefix = w.substring(0, first);

String suffix = w.substring(last+1);

String sub = getSubstitute(w.substring(first+1, last));

int index = used.indexOf(sub);

int usedornot = 1;

int counter = 0;

while (usedornot == 1){

if(index == - 1){

used.add(sub);

counter =+1;

myFreqs.add(counter);

usedornot = 0;

}else{

sub = getSubstitute(w.substring(first+1, last));

index = used.indexOf(sub);

}}

return prefix + sub+suffix;

}

private void printOut(String s, int lineWidth){

int charsWritten = 0;

for(String w : s.split("\\s+")){

if (charsWritten + w.length() > lineWidth){

System.out.println();

charsWritten = 0;

}

System.out.print(w+" ");

charsWritten += w.length() + 1;

}

}

private String fromTemplate(String source){

String story = "";

if (source.startsWith("http")) {

URLResource resource = new URLResource(source);

for(String word : resource.words()){

story = story + processWord(word) + " ";

}

}

else {

FileResource resource = new FileResource(source);

for(String word : resource.words()){

story = story + processWord(word) + " ";

}

}

return story;

}

private ArrayList<String> readIt(String source){

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

if (source.startsWith("http")) {

URLResource resource = new URLResource(source);

for(String line : resource.lines()){

list.add(line);

}

}

else {

FileResource resource = new FileResource(source);

for(String line : resource.lines()){

list.add(line);

}

}

return list;

}

public void findNoOfWordsReplaced(){

int count = myFreqs.size();

System.out.println("Total no of words replaced : " + myFreqs.size());

}

public void makeStory(){

System.out.println("\n");

String story = fromTemplate("data/madtemplate2.txt");

printOut(story, 60);

System.out.println("\n");

findNoOfWordsReplaced();

}

}

Output :

This is a slippery story about how a blue lion became a

orange polar bear. Once upon a time, about 65 centuries ago,

angry, gigantic pangolins roamed the earth. One of them was

named Lance. This horse was alone in the world. Then it

became a green fox living in Mexico. This animal loved to

pout and scream. In the morning it would eat a black pear,

and later eat a furious apple for a snack.

Total no of words replaced : 21

import java.io.\*;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

**public** **class** wordsInFiles {

**private** HashMap<String, ArrayList<String>>wordsMap;

**public** wordsInFiles(){

wordsMap = **new** HashMap<String, ArrayList<String>>();

}

//This method add all the words from f into the map.

// If a word is not in the map, then create a new ArrayList of type String with this word, and have

// the word map to this ArrayList.

// If a word is already in the map, then add the current filename to its ArrayList

**private** **void** addWordsInFile(File f){

FileResource fileResource = **new** FileResource(f.toString());

String fileName = f.getName();

**for** (String word : fileResource.words()) {

**if** (!wordsMap.containsKey(word)) {

ArrayList<String> newList = **new** ArrayList<String>();

newList.add(fileName);

wordsMap.put(word, newList);

} **else** {

//if it does already contain the key, it gets the current value (the arraylist which will have //some things in), adds //the thing you want to add to the arraylist, and then puts the key and //the updated value in the map. Each key in //the hashmap will only occur once - this doesn't //add another key, it updates the existing value. Each key in the //hashmap will still be unique.

ArrayList<String> newList = wordsMap.get(word);

newList.add(fileName);

wordsMap.put(word, newList);

}}}

**public** **void** buildWordsFileMap(){

wordsMap.clear();

DirectoryResource res = **new** DirectoryResource();

**for**(File f : res.selectedFiles()){

addWordsInFile(f);

}

}

// This method returns the maximum number of files any word appears in, considering all words

// from a group of files. There are 4 files. Two words appear in three of the files.

// So, the maxNumber would return 3.

**public** **int** maxNumber(){

//System.out.print("The greatest number of files a word appears in is");

**int** max = 0;

**for**(String f : wordsMap.keySet()){

ArrayList<String>currentFile = wordsMap.get(f);

**int** count = currentFile.size();

**if** ( max < count){

max = count;

}

}

**return** max;

}

//This method returns an ArrayList of words that appear in exactly number files.

// the call wordsInNumFiles(2) would return an ArrayList with the words "love","are" and "dogs"

**private** ArrayList<String> wordsInNumFiles(**int** number){

ArrayList<String>wordList = **new** ArrayList<String>();

**for** (String key : wordsMap.keySet()){

ArrayList<String>currentList = wordsMap.get(key);

**int** currentFilecount = currentList.size();

**if** (currentFilecount == number){

wordList.add(key);

}

}

**return** wordList;

}

// This method prints the names of the files this word appears in, one filename per line.

**public** **void** printFiles(String words){

System.***out***.println("The files contain "+ words +" are(is): \t");

ArrayList<String> fileNames = wordsMap.get(words);

**for** (**int** index = 0; index < fileNames.size(); index++) {

System.***out***.println(fileNames.get(index));

}

}

**public** **void** tester(){

buildWordsFileMap();

**int** max = maxNumber();

ArrayList<String> list = wordsInNumFiles(max);

System.***out***.println("The greatest number of files a word appears in is "+max+", and there are "+list.size()+ " such words: ");

**for** (**int** k = 0; k < list.size(); k++) {

System.***out***.println(list.get(k)+" ");

}

System.***out***.println("\t");

**for** (**int** k = 0; k < list.size(); k++) {

printFiles(list.get(k));

}}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

wordsInFiles wf = **new** wordsInFiles();

wf.tester();

}

}

Output:

The greatest number of files a word appears in is 3, and there are 2 such words:

cats

and

The files contain cats are(is):

brief1.txt

brief3.txt

brief4.txt

The files contain and are(is):

brief1.txt

brief3.txt

brief4.txt

Count of all words that appear in all files: 17

import edu.duke.\*;

import java.util.\*;

public class GladLib {

private HashMap<String, ArrayList<String>>GladMap;

private Random myRandom;

private ArrayList<String>used;

private ArrayList<Integer>freq;

private ArrayList<String>usedCategory;

private static String dataSourceURL = "http://dukelearntoprogram.com/course3/data";

private static String dataSourceDirectory = "data";

public GladLib(){

GladMap = new HashMap<String, ArrayList<String>>();

initializeFromSource(dataSourceDirectory);

used = new ArrayList<String>();

freq = new ArrayList<Integer>();

usedCategory = new ArrayList<String>();

myRandom = new Random();

}

public GladLib(String source){

GladMap = new HashMap<String, ArrayList<String>>();

initializeFromSource(source);

used = new ArrayList<String>();

freq = new ArrayList<Integer>();

usedCategory = new ArrayList<String>();

myRandom = new Random();

}

private void initializeFromSource(String source) {

String [] labels = {"adjective", "noun", "color", "timeframe", "verb", "fruit", "country", "animal", "name"};

for (String s : labels){

ArrayList <String> list = readIt(source + "/" + s + ".txt");

GladMap.put(s, list);

}

}

private String randomFrom(ArrayList<String> source){

int index = myRandom.nextInt(source.size());

return source.get(index);

}

private String getSubstitute(String label) {

if (GladMap.containsKey(label)){

if (!usedCategory.contains(label))

usedCategory.add(label);

return randomFrom(GladMap.get(label));

}

if (label.equals("number")){

return ""+myRandom.nextInt(50)+5;

}

else{

if (!label.equals("number")){

return randomFrom(GladMap.get(label));

}}

return "\*\*UNKNOWN\*\*";

}

private String processWord(String w){

int first = w.indexOf("<");

int last = w.indexOf(">", first);

if(first == 1 || last == -1){

return w;

}

String prefix = w.substring(0, first);

String suffix = w.substring(last+1);

String sub = getSubstitute(w.substring(first+1, last));

int index = used.indexOf(sub);

int usedornot = 1;

while (usedornot == 1){

if(index == - 1){

used.add(sub);

usedornot = 0;

}else{

sub = getSubstitute(w.substring(first+1, last));

index = used.indexOf(sub);

}}

return prefix + sub+suffix;

}

private void printOut(String s, int lineWidth){

int charsWritten = 0;

for(String w : s.split("\\s+")){

if (charsWritten + w.length() > lineWidth){

System.out.println();

charsWritten = 0;

}

System.out.print(w+" ");

charsWritten += w.length() + 1;

}

}

private String fromTemplate(String source){

String story = "";

if (source.startsWith("http")) {

URLResource resource = new URLResource(source);

for(String word : resource.words()){

story = story + processWord(word) + " ";

}

}

else {

FileResource resource = new FileResource(source);

for(String word : resource.words()){

story = story + processWord(word) + " ";

}

}

return story;

}

private ArrayList<String> readIt(String source){

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

if (source.startsWith("http")) {

URLResource resource = new URLResource(source);

for(String line : resource.lines()){

list.add(line);

}

}

else {

FileResource resource = new FileResource(source);

for(String line : resource.lines()){

list.add(line);

}

}

return list;

}

private int totalWordsInMap() {

int sum = 0;

for (String word: GladMap.keySet()) {

sum += GladMap.get(word).size();

}

return sum;

}

private int totalWordsConsidered() {

int sum = 0;

for (int k = 0; k < usedCategory.size(); k++) {

sum += GladMap.get(usedCategory.get(k)).size();

}

return sum;

}

public void makeStory(){

System.out.println("\n");

String story = fromTemplate("data/madtemplate2.txt");

printOut(story, 60);

System.out.print("\n");

System.out.print("The total no of words that were possible to pick from : " + totalWordsInMap());

System.out.print("\n");

System.out.print("The total no of words that were used in categories : " + totalWordsConsidered());

}

}

Output:

This is a furious story about how a orange armadillo became

a black lion. Once upon a time, about 235 months ago,

slippery, gigantic polar bears roamed the earth. One of them

was named Jermaine. This pangolin was alone in the world.

Then it became a yellow tiger living in Switzerland. This

animal loved to pout and scream. In the morning it would eat

a pink plum, and later eat a funny strawberry for a snack.

The total no of words that were possible to pick from : 70

The total no of words that were used in categories : 64

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

public class LogAnalyzer {

private ArrayList<LogEntry> records;

public LogAnalyzer() {

// complete constructor

// records.clear();

records = new ArrayList<LogEntry>();

}

public void readFile(String filename) {

// complete method

FileResource res = new FileResource("C:\\Users\\Karen Goh Seow Hui\\ELunar\\Assignment5\\src\\LogEntry\\short-test\_log");

for(String s : res.lines()){

LogEntry logentry = WebLogParser.parseEntry(s);

records.add(logentry);

}

}

public int countUniqueiPS(){ // note: by using int method, you can get the ArrayList of uniqueIps and the size as opp to just use the ArrayList<String> method

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

for(LogEntry le : records){

String ipAddr = le.getIpAddress();

if(!uniqueIps.contains(ipAddr)){

uniqueIps.add(ipAddr);

}

}

return uniqueIps.size();

}

public void printAllHigherThanNum(int num){

int count = 0;

// ArrayList<String>uniqueIps = new ArrayList<String>();

for(LogEntry le : records){

int ipAddr = le.getStatusCode();

if (ipAddr > num)

System.out.println(ipAddr);

//System.out.println(s.toString());

}

}

public ArrayList<String>uniqueIPVisitsOnDay(String someday){

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

for(LogEntry le : records){

Date testsomedayParsed = le.getAccessTime();

SimpleDateFormat formatter = new SimpleDateFormat("MMM dd");

String daytest = formatter.format(testsomedayParsed);

String somedaytoString = daytest.toString();

int index = somedaytoString.indexOf(someday);

if (index != -1) {

// if(!testdate.contains(somedaytoString)){

testdate.add(testsomedayParsed.toString());

// if(!testdate.contains(someday)){

// testdate.add(somedaytoString);

}}

return testdate;

}

public int countUniqueIPsInRange(int low, int high){

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

for(LogEntry le : records){

int ipAddr = le.getStatusCode();

if (ipAddr >= low && ipAddr <= high){

String uniqueipAddr = le.getIpAddress();

if(!uniqueIps.contains(uniqueipAddr)){

uniqueIps.add(uniqueipAddr);

}

}}

return uniqueIps.size();

}

public void printAll() {

for (LogEntry le : records) {

System.out.println(le);

}

}

public static void main (String [] args){

LogAnalyzer la = new LogAnalyzer();

la.readFile("C:\\Users\\Karen Goh Seow Hui\\ELunar\\Assignment5\\src\\LogEntry\\short-test\_log");

int uniqueIps = la.countUniqueiPS();

System.out.print("There are " + uniqueIps + " unique IPs");

System.out.print("\n");

la.printAllHigherThanNum(200);

System.out.print("\n");

// la.printAll();

// System.out.println(la.uniqueIPVisitsOnDay("Sep 14"));

System.out.println(la.countUniqueIPsInRange(200,299));

System.out.println(la.countUniqueIPsInRange(300,399));

}

}

Output:

There are 4 unique IPs

302

304

302

4

2

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.logging.SimpleFormatter;

**public** **class** LogAnalyzer {

**private** **static** ArrayList<LogEntry> *records*;

**public** LogAnalyzer() {

// complete constructor

// records.clear();

*records* = **new** ArrayList<LogEntry>();

}

**public** **void** readFile(String filename) {

// complete method

FileResource res = **new** FileResource("C:\\Users\\Karen Goh Seow Hui\\ELunar\\Assignment9\\src\\LogEntry\\weblog3-short\_log");

**for**(String s : res.lines()){

LogEntry logentry = WebLogParser.*parseEntry*(s);

*records*.add(logentry);

}

}

**public** **int** countUniqueiPS(){ // note: by using int method, you can get the ArrayList of uniqueIps and the size as opp to just use the ArrayList<String> method

ArrayList<String>uniqueIps = **new** ArrayList<String>();

**for**(LogEntry le : *records*){

String ipAddr = le.getIpAddress();

**if**(!uniqueIps.contains(ipAddr)){

uniqueIps.add(ipAddr);

}

}

**return** uniqueIps.size();

}

**public** **void** printAllHigherThanNum(**int** num){

**for**(LogEntry le : *records*){

**int** ipAddr = le.getStatusCode();

**if** (ipAddr > num)

System.***out***.println(ipAddr);

}

}

**public** ArrayList<String>uniqueIPVisitsOnDay(String someday){

ArrayList<String>testdate = **new** ArrayList<String>();

**for**(LogEntry le : *records*){

Date testsomedayParsed = le.getAccessTime();

SimpleDateFormat formatter = **new** SimpleDateFormat("MMM dd");

String daytest = formatter.format(testsomedayParsed);

String somedaytoString = daytest.toString();

**int** index = somedaytoString.indexOf(someday);

**if** (index != -1) {

testdate.add(somedaytoString);

}}

**return** testdate;

}

**public** **int** countUniqueIPsInRange(**int** low, **int** high){

ArrayList<String>uniqueIps = **new** ArrayList<String>();

**for**(LogEntry le : *records*){

**int** ipAddr = le.getStatusCode();

**if** (ipAddr >= low && ipAddr <= high){

String uniqueipAddr = le.getIpAddress();

**if**(!uniqueIps.contains(uniqueipAddr)){

uniqueIps.add(uniqueipAddr);

}

}}

**return** uniqueIps.size();

}

**public** HashMap<String, Integer>countVisitsPerIP(){

HashMap<String, Integer>uniqueIps = **new** HashMap<String, Integer>();

**for**(LogEntry le : *records*){

String ip = le.getIpAddress();

**if** (!uniqueIps.containsKey(ip)){

uniqueIps.put(ip, 1);

}**else**{

uniqueIps.put(ip, uniqueIps.get(ip)+1);

}}

**return** uniqueIps;

}

//alternate solution

**public** **int** countUniqueIps(){

HashMap<String, Integer>uniqueIps = countVisitsPerIP();

**return** uniqueIps.size();

}

**public** **int** mostNumberVisitsByIP(HashMap<String, Integer> uniqueIps){//since Hashmap is already created at above method, no need to new HashMap

**int** count = 0;

**int** maxIP = 0;

//for(LogEntry le : records){

**for** (String s : uniqueIps.keySet()){

count = uniqueIps.get(s);

//maxIP+=1;

**if** (maxIP < count)

maxIP = count;

}

**return** maxIP;

}

**public** ArrayList<String> iPsMostVisits(HashMap<String, Integer> numberIPappear){//is it possible to obtain without HashMap to hold the key value in this case ?

ArrayList<String>ipList = **new** ArrayList<String>();

**int** current = 0;

**int** maxIp = mostNumberVisitsByIP(numberIPappear);

**for**(String ip : numberIPappear.keySet()){

current = numberIPappear.get(ip);

**if** (maxIp == current){

ipList.add(ip);

}

}

**return** ipList;

}

//Date is String and ArrayList is IP address

**public** HashMap<String, ArrayList<String>>iPsForDays(){

HashMap<String, ArrayList<String>>ipDay = **new** HashMap<String, ArrayList<String>>();

**for**(LogEntry le : *records*){

Date ipdate = le.getAccessTime();

SimpleDateFormat formatter = **new** SimpleDateFormat("MMM dd");

String day = formatter.format(ipdate);

String daytoString = day.toString();

// System.out.println("the dates are " + daytoString);

String ip = le.getIpAddress();

**if**(!ipDay.containsKey(daytoString)){

//Note : there is no need to create a new ArrayList

// ArrayList<String> newList = new ArrayList<String>();

// ipDay.put(daytoString, new ArrayList<String>());

// newList.add(ip);

ipDay.put(daytoString, **new** ArrayList<String>());

}

ipDay.get(daytoString).add(ip);// to display the date and correspondence ip values accordingly

// ipDay.get(daytoString, newList.add(ip));

}

**return** ipDay;

}

**public** String dayWithMostIPVisits(HashMap<String, ArrayList<String>>ipDay){

String maxday = "";

**int** maxdaycount = 0;

**int** current;

**for**(String s : ipDay.keySet()){

current = ipDay.get(s).size(); // note: HashMap is not Integer, hence you have to add .size();

**if** (maxdaycount < current){

maxdaycount = current;

maxday = s;

}}

**return** maxday;

}

**public** ArrayList<String>iPsWithMostVisitsOnDay(HashMap<String, ArrayList<String>> eachDay, String day){

ArrayList<String>iPs = **new** ArrayList<String>();

HashMap<String, Integer>EachIp = **new** HashMap<String, Integer>();

ArrayList<String>MostiPsday = **new** ArrayList<String>();

**for**(String time : eachDay.keySet()){

**if** (time.equals(day)){

iPs = eachDay.get(time);

//System.out.println("to test" + iPs);

}}

**for**(**int** i = 0; i < iPs.size(); i++){

**if**(!EachIp.containsKey(iPs.get(i))){

EachIp.put(iPs.get(i),1);

}

**else**

EachIp.put(iPs.get(i), EachIp.get(iPs.get(i))+ 1);

}

MostiPsday = iPsMostVisits(EachIp);

**return** MostiPsday;

}

**public** **void** testmostNumberVisitByIP(){

LogAnalyzer la = **new** LogAnalyzer();

la.readFile("C:\\Users\\Karen Goh Seow Hui\\ELunar\\Assignment9\\src\\LogEntry\\weblog3-short\_log");

HashMap<String, Integer>max = la.countVisitsPerIP();

**int** high = la.mostNumberVisitsByIP(max);

System.***out***.println(high);

}

**public** **void** testIPMostVisit(){

LogAnalyzer la = **new** LogAnalyzer();

la.readFile("C:\\Users\\Karen Goh Seow Hui\\ELunar\\Assignment9\\src\\LogEntry\\weblog3-short\_log");

HashMap<String, Integer>max = la.countVisitsPerIP();

//ArrayList<String> iPsMostVisits = la.iPsMostVisits(max);

//Note: here you can name it as result; not necessarity iPsMostVisits

ArrayList<String> result = la.iPsMostVisits(max);

System.***out***.println(result);

System.***out***.println(result.get(0)); // to find out the first most visit IP

}

**public** **void** testiPsForDays(){

LogAnalyzer la = **new** LogAnalyzer();

la.readFile("C:\\Users\\Karen Goh Seow Hui\\ELunar\\Assignment9\\src\\LogEntry\\weblog3-short\_log");

HashMap<String, ArrayList<String>>ipAndDate = la.iPsForDays();

System.***out***.println(ipAndDate);

// System.out.println(result.get(0)); // to find out the first most visit IP

}

**public** **void** testdayWithMostIPVisit(){

LogAnalyzer la = **new** LogAnalyzer();

la.readFile("C:\\Users\\Karen Goh Seow Hui\\ELunar\\Assignment9\\src\\LogEntry\\weblog3-short\_log");

HashMap<String, ArrayList<String>>maxDay = la.iPsForDays();

String mosttime = la.dayWithMostIPVisits(maxDay);

System.***out***.println(mosttime);

}

**public** **void** testiPsWithMostVisitsOnDay(){

LogAnalyzer la = **new** LogAnalyzer();

la.readFile("C:\\Users\\Karen Goh Seow Hui\\ELunar\\Assignment9\\src\\LogEntry\\weblog3-short\_log");

HashMap<String, ArrayList<String>>maxDay = la.iPsForDays();

ArrayList<String> result = la.iPsWithMostVisitsOnDay(maxDay, "Sep 30");

**for**(**int** k = 0; k < result.size(); k++){

System.***out***.println(result.get(k));

}

}

**public** **void** printAll() {

ArrayList<LogEntry>records1 = **new** ArrayList<LogEntry>();

**for** (LogEntry le : records1) {

System.***out***.println(le);

}

}

**public** **static** **void** main (String [] args){

LogAnalyzer la = **new** LogAnalyzer();

la.readFile("C:\\Users\\Karen Goh Seow Hui\\ELunar\\Assignment9\\src\\LogEntry\\weblog3-short\_log");

**int** uniqueIps = la.countUniqueiPS();

System.***out***.print("There are " + uniqueIps + " unique IPs");

System.***out***.print("\n");

la.printAllHigherThanNum(200);

System.***out***.print("\n");

la.printAll();

// HashMap<String, Integer>max = new HashMap<String, Integer>();

//// // for (LogEntry le : records){

// HashMap<String, Integer>maxcount = LogAnalyzer.countVisitsPerIP();

//System.out.println(maxcount);

// System.out.println(la.countUniqueIps());//print out the no of different Ip in the log

la.testmostNumberVisitByIP();

la.testIPMostVisit();

la.testiPsForDays();

la.testdayWithMostIPVisit();

//la.testiPsWithMostVisitsOnDay();

la.testiPsWithMostVisitsOnDay();

}}

Output:

There are 4 unique IPs

404

404

3

[61.15.121.171, 84.133.195.161]

61.15.121.171

{Sep 14=[84.133.195.161], Sep 21=[84.189.158.117, 61.15.121.171, 84.133.195.161, 84.133.195.161], Sep 30=[84.189.158.117, 61.15.121.171, 61.15.121.171, 177.4.40.87, 177.4.40.87]}

Sep 30

61.15.121.171

177.4.40.87

Duke Source files

package edu.duke;

import java.io.File;

import java.util.ArrayList;

import java.util.Arrays;

/\*\*

\* The <code>DirectoryResource</code> class allows the user to choose one or more files from a directory (or folder) with a file selection dialog box by using the method

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

\*

\* Example usage:

\*

\* <PRE>

\* DirectoryResource dr = new DirectoryResource();

\* for (File f : dr.selectedFiles()) {

\* ImageResource ir = new ImageResource(f);

\* ir.draw();

\* }

\* </PRE>

\* 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 DirectoryResource { // create a DirectoryResource object

public DirectoryResource () {

// do nothing

}

/\*\*

\* Open a file selection dialog box to allow the user to navigate to a directory and select one or more files from the chosen directory (or folder).

\*

\* The file selection dialog box opened starts in the current project folder.

\*

\* @return an <code>Iterable</code> that accesses the chosen files one at a time

\*/

public Iterable<File> selectedFiles () {

File[] files = FileSelector.selectFiles();

// guaranteed to have at least one item

if (files[0] == null) {

// return empty list rather than null, so others can throw the exception if needed

return new ArrayList<File>();

}

else {

return Arrays.asList(files);

}

}

}

package edu.duke;

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 FileResource class represents a file and allows access to its contents a line at a time, using the method lines, or a word at a time, using the method words. These strings can then be iterated over using a for loop.

\*

\* Example usage:

\*

\* FileResource fr = new FileResource();

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

\* // print or process

\*

\* If each line of the file represents separated data values, because its a CSV file, then the user can get a getCSVParser object to access that data more directly, using one of the getCSVParser methods.

\* Example CSV usage:

\*

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

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

\* // print or process fields in record

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

\* // other processing

\* }

\* @author Duke Software Team

\*/

public class FileResource {

private String myPath;

private String mySource;

private File mySaveFile;

/\*\*

\* Create a FileResource 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 FileResource object that opens a file represented by the File object passed as a parameter.

\* Useful, for example, when used in conjunction with the DirectoryResource 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 FileResource 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 FileResource 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

\* write to add new strings to it, pass true 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 FileResource 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

\* write to add new strings to it, pass true 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 DirectoryResource 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 FileResource 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

\* write to add new strings to it, pass true 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 Iterable 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 Iterable 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 String that is the contents of the open file

\*/

public String asString () {

return mySource;

}

/\*\*

\* Returns a CSVParser 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 CSVParser 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 CSVParser 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 CSVParser 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 CSVParser 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 CSVParser 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 delimiter.length() != 1

\*/

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

Iterator is returned.

\*

\* @param parser the CSVParser that has been created for this file

\* @return an Iterable 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);

}

}

}

package edu.duke;

import java.io.File;

import javax.swing.JFileChooser;

import javax.swing.JOptionPane;

import javax.swing.SwingUtilities;

import javax.swing.filechooser.FileFilter;

/\*\*

\* This utility class creates a thread safe file dialog box for loading and

\* saving files.

\*

\* @author Duke Software Team

\*

\*/

class FileSelector {

// result of selection

private static File[] ourFiles;

// BUGBUG: I think this is the right behavior, remembers where user left it last

private static JFileChooser ourChooser = new JFileChooser();

static {

ourChooser.setFileSelectionMode(JFileChooser.FILES\_ONLY);

ourChooser.setCurrentDirectory(new File(System.getProperty("user.dir")));

}

/\*\*

\* Pops up a dialog box to select only one file.

\*

\* @return

\*/

public static File selectFile () {

// guaranteed to have one element, though it may be null

return selectFiles(null, false, true)[0];

}

/\*\*

\* Pops up a dialog box to select only one file with given extensions.

\*/

public static File selectFile (String[] extensionAccepted) {

// guaranteed to have one element, though it may be null

return selectFiles(extensionAccepted, false, true)[0];

}

/\*\*

\* Pops up a dialog box to select multiple files.

\*/

public static File[] selectFiles () {

return selectFiles(null, true, true);

}

/\*\*

\* Pops up a dialog box to select multiple files with given extensions.

\* @return

\*/

public static File[] selectFiles (String[] extensionAccepted) {

return selectFiles(extensionAccepted, true, true);

}

/\*\*

\* Pops up a dialog box to save file with any extension.

\*/

public static File saveFile () {

// guaranteed to have one element, though it may be null

return selectFiles(null, false, false)[0];

}

/\*\*

\* Pops up a dialog box to save file with given extensions.

\*/

public static File saveFile (String[] extensionAccepted) {

// guaranteed to have one element, though it may be null

return selectFiles(extensionAccepted, false, false)[0];

}

// BUGBUG: one general function, but lots of booleans :(

private static File[] selectFiles (String[] extensionAccepted,

final boolean allowMultiple,

final boolean openForRead) {

ourChooser.setMultiSelectionEnabled(allowMultiple);

ourChooser.setFileFilter(new ChooserFilter(extensionAccepted));

try {

ourFiles = null;

SwingUtilities.invokeAndWait(new Runnable() {

@Override

public void run () {

int result = 0;

if (openForRead) {

result = ourChooser.showOpenDialog(null);

}

else {

result = ourChooser.showSaveDialog(null);

}

if (result == JFileChooser.CANCEL\_OPTION) {

ourFiles = new File[] { null };

} else {

try {

if (allowMultiple) {

ourFiles = ourChooser.getSelectedFiles();

} else {

ourFiles = new File[] { ourChooser.getSelectedFile() };

}

} catch (Exception e) {

JOptionPane.showMessageDialog(null, e.toString());

}

}

}

});

return ourFiles;

} catch (Exception e) {

// it is still an exception, just not one required to be handled

throw new RuntimeException(e);

}

}

// This class implements a filter for image file names.

static class ChooserFilter extends FileFilter {

private String myExtensions;

public ChooserFilter (String[] extensionsAccepted) {

if (extensionsAccepted != null) {

myExtensions = String.format("(?i).\*\\.(%s)", String.join("|", extensionsAccepted));

}

}

@Override

public boolean accept (File f) {

if (myExtensions != null) {

return f.getName().matches(myExtensions) || f.isDirectory();

} else {

return true;

}

}

@Override

public String getDescription () {

// BUGBUG: useful?

return "Files";

}

}

}

package edu.duke;

import java.util.Iterator;

\* This utility class allows multiple classes to iterate over a text source in

\* multiple ways.

\* It also serves to show how to implement an iterator.

\* @author Duke Software Team

\*

class TextIterable implements Iterable<String> {

private String[] myStrings;

/ \* Create from a given string. /

public TextIterable (String source, String regexp) {

myStrings = source.split(regexp);

}

/\*\*

\* @see java.lang.Iterator

\*/

@Override

public Iterator<String> iterator () {

return new Iterator<String>() {

private int myCount = 0;

@Override

public boolean hasNext () {

return myCount < myStrings.length;

}

@Override

public String next () {

String s = myStrings[myCount];

myCount++;

return s;

}

};

}

}

package edu.duke;

import java.util.List;

import java.util.ArrayList;

/\*\*

\* The RangeResource class represents a range of integer (or whole) numbers and allows access to these numbers using the method sequence. The integers can then be iterated over using a for loop

\*

\* Example usage:

\*

\* // this prints the square number sequence for the first 9 numbers:

\* // 1, 4, 9, 16, 25, 36, 49, 64, 81

\* RangeResource rr = new RangeResource(1, 10);

\* for (int value : rr.sequence()) {

\* System.out.println(value \* value);

\* }

\* Another example usage:

\*

\* // this prints the odd numbers in decreasing order within the given range:

\* // 37, 35, 33, 31, 29, 27, 25, 23

\* RangeResource rr = new RangeResource(37, 21, -2);

\* for (int value : rr.sequence()) {

\* System.out.println(value);

\* }

\* @author Duke Software Team

\*/

public class RangeResource {

private int myStart;

private int myEnd;

private int myIncrement;

private List<Integer> myValues;

/\*\*

\* Create a RangeResource object, starting at 0 and going up to but not including end, [0 - end), that increments by 1.

\*

\* @param end when to stop the range, not included as one of the values

\* @throws exception if the end is negative

\*/

public RangeResource (int end) {

this(0, end, 1);

}

/\*\*

\* Create a RangeResource object, starting at start and going up to but not including end, [start - end), that increments by 1.

\*

\* @param start the first value in the range, included as one of the values

\* @param end when to stop the range, not included as one of the values

\* @throws exception if the end is less than the start

\*/

public RangeResource (int start, int end) {

this(start, end, 1);

}

/\*\*

\* Create a RangeResource object, starting at start and going up to but not including end, [start - end), that increments by the amount passed as a parameter.

\*

\* @param start the first value in the range, included as one of the values

\* @param end when to stop the range, not included as one of the values

\* @param increment how much to add to get the next value in the range's sequence

\* @throws exception if increment is negative when the end is greater than the start

\* @throws exception if increment is positive when the end is less than the start

\* @throws exception if increment is 0

\*/

public RangeResource (int start, int end, int increment) {

if (increment == 0) {

throw new ResourceException("RangeResource: invalid increment, cannot be 0");

}

if (end < start && increment > 0) {

throw new ResourceException("RangeResource: invalid increment, cannot be positive when end (" + end + ") is less than start (" + start + ")");

}

if (end > start && increment < 0) {

throw new ResourceException("RangeResource: invalid increment, cannot be negative when end (" + end + ") is greater than start (" + start + ")");

}

myStart = start;

myEnd = end;

myIncrement = increment;

myValues = makeValues(start, end, increment);

}

/\*\*

\* Create an RangeResource object that is a copy of another range.

\*

\* @param other the original range being copied

\*/

public RangeResource (RangeResource other) {

this(other.myStart, other.myEnd, other.myIncrement);

}

/\*\*

\* Return string representation of this range, with each value in the sequence separated by a comma.

\*

\* @return a String containing all the values in the range

\*/

@Override

public String toString () {

// System.out.println("RANGE: [" + myStart + ".." + myEnd + ") by " + myIncrement);

return myValues.toString();

// old iteration way

// StringBuilder result = new StringBuilder();

// result.append("[ ");

// for (int k : this.sequence()) {

// result.append(k + " ");

// }

// result.append("]");

// return result.toString();

}

/\*\*

\* Allow access to the numbers in this range one at a time.

\*

\* @return an Iterable that will allow access to each number in this range

\*/

public Iterable<Integer> sequence () {

return myValues;

// left in case we want to show another example of an iterator

// return new Iterable<Integer>() {

// @Override

// public Iterator<Integer> iterator () {

// return new Iterator<Integer>() {

// private int place = myStart;

// @Override

// public boolean hasNext () {

// if (myIncrement > 0) {

// return place < myEnd;

// }

// else {

// return place > myEnd;

// }

// }

// @Override

// public Integer next () {

// int result = place;

// place += myIncrement;

// return result;

// }

// };

// }

// };

}

// generate the values in the range (much simpler than building an Iterator)

private List<Integer> makeValues (int start, int end, int increment) {

List<Integer> result = new ArrayList<Integer>();

while (true) {

if (increment > 0 && start >= end) {

break;

}

else if (increment < 0 && start <= end) {

break;

}

result.add(start);

start += increment;

}

return result;

}

}

package edu.duke;

import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.Reader;

import java.io.StringReader;

import java.net.URL;

import org.apache.commons.csv.CSVFormat;

import org.apache.commons.csv.CSVParser;

/\*\*

\* The URLResource class opens a connection to a URL and allows access to the contents

\* of the web page a line at a time, using the method lines, or a word at a time, using the method words. These strings can then be iterated over using a for loop.

\*

\* Example usage:

\*

\* URLResource ur = new URLResource("http://www.dukelearntoprogram.com/");

\* for (String s : ur.lines()) {

\* // print or process s

\* }

\* If each line of the web page represents separated data values, because its a CSV file, then the user can get a getCSVParser object to access that data more directly, using one of the getCSVParser methods.

\*

\* Example CSV usage:

\*

\* URLResource ur = new URLResource("http://www.dukelearntoprogram.com/course2/java/food.csv");

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

\* // print or process fields in record

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

\* // other processing

\* }

\* @author Duke Software Team

public class URLResource {

private String myPath;

private String mySource;

/\*\*

\* Create a URLResource object bound to the web page whose URL is given as the parameter.

\*

\* Constructing the object opens a connection and reads the contents of the web page.

\*

\* @param name is the name of the URL, it must start with "http" or "https"

\* @throws exception if the URL does not start with "http" or "https"

\*/

public URLResource (String name) {

if (name.startsWith("http://") || name.startsWith("https://")) {

try {

mySource = initFromStream(new URL(name).openStream());

myPath = name;

}

catch (Exception e) {

throw new ResourceException("URLResource: unable to load URL with name " + name, e);

}

}

else {

throw new ResourceException("URLResource: name must start with http:// or https://" + name);

}

}

/\*\*

\* Allow access to open web page one line at a time.

\*

\* @return an Iterable that allows access one line at a time

\*/

public Iterable<String> lines () {

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

}

/\*\*

\* Allow access to this open web page 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 Iterable that allows access one word at a time

\*/

public Iterable<String> words () {

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

}

/\*\*

\* Return entire open web page as one string.

\*

\* @return a String that is the contents of the open web page

\*/

public String asString () {

return mySource;

}

/\*\*

\* Returns a CSVParser object to access the contents of an open web page.

\*

\* Each line of the web page should be formatted as data separated by commas and with a header row to describe the column names.

\*

\* @return a CSVParser that can provide access to the records in the web page one at a time

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

\*/

public CSVParser getCSVParser () {

return getCSVParser(true);

}

/\*\*

\* Returns a CSVParser object to access the contents of an open web page, possibly

\* without a header row.

\*

\* Each line of the web page 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 CSVParser that can provide access to the records in the web page one

at a time

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

\*/

public CSVParser getCSVParser (boolean withHeader) {

return getCSVParser(withHeader, ",");

}

/\*\*

\* Returns a CSVParser object to access the contents of an open web page, possibly

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

\*

\* Each line of the web page 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 web page one at a time

\* @throws exception if this web page 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("URLResource: 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("URLResource: 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

Iterator is returned.

\*

\* @param parser the CSVParser that has been created for this web page

\* @return an Iterable that allows access one header name at a time

\*/

public Iterable<String> getCSVHeaders (CSVParser parser) {

return parser.getHeaderMap().keySet();

}

// 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("URLResource: error encountered reading " + myPath, e);

}

finally {

try {

if (buff != null) {

buff.close();

}

}

catch (Exception e) {

// should never happen

}

}

}

}

package edu.duke;

/\*\*

\* A general exception for Resource objects.

Note, this is a RuntimeException so that it does not HAVE to be caught,

but still gives useful information about the error.

\* @author Duke Software Team

\*/

public class ResourceException extends RuntimeException {

private static final long serialVersionUID = 1L;

public ResourceException (String message) {

super(message);

}

public ResourceException (String message, Throwable cause) {

super(message, cause);

}

}

package edu.duke;

/\*\*

\* The <code>Pixel</code> class represents a color as its component values of

\* red, green, blue, as well as alpha (for transparency).

\*

\* <P>

\* Each of the component values of a pixel must have a value between 0 and 255.

\* If a value is given outside that range, it is changed to be within that range.

\* As such, a negative value would be set to 0 and a value greater than 255 would

\* be set to 255.

\*

\* <P>

\* Example usage:

\*

\* <pre>

\* ImageResource image = new ImageResource();

\* for (Pixel p : image.pixels()) {

\* int red = p.getRed();

\* int green = p.getGreen();

\* int blue = p.getBlue();

\* int average = (red + green + blue) / 3;

\* p.setRed(average);

\* p.setGreen(average);

\* p.setBlue(average);

\* }

\* image.draw();

\* </pre>

\*

\* <P>

\* This is open-source software released under the terms of the GPL

\* (http://www.gnu.org/licenses/gpl.html).

\*/

public class Pixel {

static final int MAX\_VALUE = 255;

private int alpha = MAX\_VALUE, red = 0, green = 0, blue = 0;

private int myX;

private int myY;

/\*\*

\* Creates a Pixel from an integer value.

\*

\* @param i the integer value representing all the color components

\* @param x the x-coordinate of this pixel in the image

\* @param y the y-coordinate of this pixel in the image

\*/

Pixel (int i, int x, int y) {

myX = x;

myY = y;

setValue(i);

}

/\*\*

\* Creates a Pixel from RGB values and an Alpha value (for transparency).

\*

\* @param r the red value

\* @param g the green value

\* @param b the blue value

\* @param a the Alpha value

\* @param x the x-coordinate of this pixel in the image

\* @param y the y-coordinate of this pixel in the image

\*/

Pixel (int r, int g, int b, int a, int x, int y) {

red = r;

green = g;

blue = b;

alpha = a;

myX = x;

myY = y;

}

/\*\*

\* Creates a Pixel from RGB values.

\*

\* @param r the red value

\* @param g the green value

\* @param b the blue value

\* @param x the x-coordinate of this pixel in the image

\* @param y the y-coordinate of this pixel in the image

\*/

Pixel (int r, int g, int b, int x, int y) {

this(r, g, b, MAX\_VALUE, x, y);

}

/\*\*

\* Creates a new Pixel from with the same values as the other pixel passed

\* as a parameter.

\*

\* @param other another pixel

\*/

public Pixel (Pixel other) {

this(other.getRed(), other.getGreen(), other.getBlue(),

other.getAlpha(), other.getX(), other.getY());

}

/\*\*

\* Returns the pixel's x-coordinate within the image.

\*

\* @return the x-coordinate of this pixel.

\*/

public int getX () {

return myX;

}

/\*\*

\* Returns the pixel's y-coordinate within the image.

\*

\* @return the y-coordinate of this pixel.

\*/

public int getY () {

return myY;

}

/\*\*

\* Returns the value of the pixel's red component.

\*

\* @return the pixel's red value within the range 0-255

\*/

public int getRed () {

return red;

}

/\*\*

\* Returns the value of the pixel's green component.

\*

\* @return the pixel's green value within the range 0-255

\*/

public int getGreen () {

return green;

}

/\*\*

\* Returns the value of the pixel's blue component.

\*

\* @return the pixel's blue value within the range 0-255

\*/

public int getBlue () {

return blue;

}

/\*\*

\* Returns the value of the pixel's alpha (or transparency) component.

\*

\* @return the pixel's alpha value within the range 0-255

\*/

public int getAlpha () {

return alpha;

}

/\*\*

\* Resets the value of the pixel's red component to the value passed as a parameter.

\* If it is not in the range of 0-255 it is changed to be in that range.

\*

\* @param r the red value

\*/

public void setRed (int r) {

red = clamp(r);

}

/\*\*

\* Resets the value of the pixel's green component to the value passed as a parameter.

\* If it is not in the range of 0-255 it is changed to be in that range.

\*

\* @param g the green value

\*/

public void setGreen (int g) {

green = clamp(g);

}

/\*\*

\* Resets the value of the pixel's blue component to the value passed as a parameter.

\* If it is not in the range of 0-255 it is changed to be in that range.

\*

\* @param b the blue value

\*/

public void setBlue (int b) {

blue = clamp(b);

}

/\*\*

\* Resets the value of the pixel's alpha (or transparency) component to the value passed as a parameter.

\* If it is not in the range of 0-255 it is changed to be in that range.

\*

\* @param a the alpha value

\*/

public void setAlpha (int a) {

alpha = clamp(a);

}

/\*\*

\* Returns the string representation of the of the pixel.

\*

\* @return a string containing the RGB values

\*/

public String toString () {

return "Pixel R: " + red + " G: " + green + " B: " + blue;

}

// returns the integer value of the pixel.

int getValue () {

return (alpha << 24) | (red << 16) | (green << 8) | blue;

}

// resets the pixel to an integer value.

void setValue (int pixel) {

alpha = (pixel >> 24) & 0xff;

red = (pixel >> 16) & 0xff;

green = (pixel >> 8) & 0xff;

blue = (pixel) & 0xff;

}

// clamps the given value to a valid pixel value

private int clamp (int value) {

return Math.max(0, Math.min(value, MAX\_VALUE));

}

}