**Exercises on File Handling**

1. We want to realize a program for handling new and used cars to sell.

For each car to sell, the information about the car is stored on a text file.

Realize a class Car, to handle a single car.

Each car is characterized by the following information:

model,

manufacturing year,

driven kilometers,

and price.

New cars can be distinguished by the fact that the driven kilometers are 0.

The class Car should export the following methods:

• a constructor to construct a car object, given all the data about the car as parameters;

• suitable get-methods to obtain the data about the car;

• a toString method, which does overriding of the toString method inherited from Object, and returns a string containing the data about the car;

• boolean equalTo(Car c) : that returns true if the car coincides with the car c in all of its data, and false otherwise.

Realize a class CarList, each of whose objects represents a list of cars to sell.

A CarList object does not directly store the data about the cars to sell, but maintains the name of a text file in which such data are stored, according to the following format:

model

manufacturing year

driven kilometers

price

The class CarList should export the following methods:

• CarList(String filename) : constructor with a parameter of type String, representing the name of the file in which the data about the cars to sell are stored;

• int countNewCars() : that returns the number of new cars in the list of cars to sell;

• Car mostExpensiveCar() : that returns the Car object corresponding to the most expensive car in the list of cars to sell.

• void addCar(Car c) : that adds the car c to the end of the list of cars to sell.

• void remove(Car c) : that removes from the list of cars to sell the car whose data coincides with that of c, if such a car is present, and leaves the list unchanged otherwise.

The class Car

import java.io.\*;

public class Car {

private String model;

private int year;

private int km;

private double price;

public Car(String m, int y, int k, double p) {

model = m; year = y; km = k; price = p;

}

public String toString() {

return model + ", year: " + year + ", km: " + km + ", price: " + price;

}

public String getModel() {

return model;

}

public int getYear() {

return year;

}

public int getKm() {

return km;

}

public double getPrice() {

return price;

}

public boolean equalTo(Car c) {

return this.model.equals(c.model) && this.year == c.year && this.km == c.km &&

this.price == c.price;

}

public static Car read(BufferedReader br) throws IOException {

String s = br.readLine();

if (s == null)

return null;

else

return new Car(s, Integer.parseInt(br.readLine()),Integer.parseInt(br.readLine()),

Double.parseDouble(br.readLine()));

}

}

The class CarList

import java.io.\*;

public class CarList {

private String filename;

public CarList (String fn) {

filename = fn;

}

public int countNewCars() throws IOException {

FileReader fr = new FileReader(filename);

BufferedReader br = new BufferedReader(fr);

int count = 0;

Car c = Car.read(br);

while (c != null) {

if (c.getKm() == 0)

count++;

c = Car.read(br);

}

br.close();

return count;

}

public Car mostExpensiveCar() throws IOException {

// this is the second alternative for opening a file for reading

FileInputStream is = new FileInputStream(filename);

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

double max = 0;

Car cmax = null;

Car c = Car.read(br);

while (c != null) {

if (c.getPrice() > max) {

max = c.getPrice();

cmax = c;

}

c = Car.read(br);

}

br.close();

return cmax;

}

public void addCar(Car c) throws IOException {

FileWriter fw = new FileWriter(filename, true); //open file in append mode

PrintWriter pw = new PrintWriter(fw);

pw.println(c.getModel());

pw.println(c.getYear());

pw.println(c.getKm());

pw.println(c.getPrice());

pw.close();

}

public void removeCar(Car c) throws IOException {

File f = new File(filename);

FileInputStream is = new FileInputStream(f);

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

// this is the second alternative for opening a file for writing

File ftemp = new File("cars-temporary.txt");

FileOutputStream os = new FileOutputStream(ftemp);

PrintWriter pw = new PrintWriter(os);

Car curr = Car.read(br);

while (curr != null) {

if (!curr.equalTo(c)) {

pw.println(curr.getModel());

pw.println(curr.getYear());

pw.println(curr.getKm());

pw.println(curr.getPrice());

}

curr = Car.read(br);

}

br.close();

pw.close();

ftemp.renameTo(f);

}

}

1. A PhoneDirectory holds a list of names and associated phone numbers. But a phone directory is pretty useless unless the data in the directory can be saved permanently -- that is, in a file. Write a phone directory program that keeps its list of names and phone numbers in a file. The user of the program should be able to look up a name in the directory to find the associated phone number. The user should also be able to make changes to the data in the directory. Every time the program starts up, it should read the data from the file. Before the program terminates, if the data has been changed while the program was running, the file should be re-written with the new data. Designing a user interface for the program is part of the exercise.

**Discussion**

A minimalistic solution to this problem would use the PhoneDirectory class from Section 8.4 and would provide just the two operations that are implemented in that class: look up an entry in the directory and add an entry to the directory. I decided to write an improved class that provides two other operations: delete an entry and modify the phone number for an existing entry. It also uses the "dynamic array" technique to allow an unlimited number of entries. The original class had a fixed limit of 100 entries. Finally, I decided to apply some of the ideas about correctness and robustness from Chapter 9. For example, a name should appear only once in a phone directory, but there was nothing in the original class to ensure that this would be true. In the new version, an attempt to add a second entry for a name that already exists will cause an exception to be thrown. Any attempt to use a null string as a name or phone number will generate an exception.

(On the other hand, my class is still not suitable for use in a real phone directory application, for several reasons. The linear search used to look up a phone number is too inefficient when applied to large arrays. The representation of an entry in the phone directory as two strings, giving the name and the number, is too limited. There is no way to view part of the directory, such as all the names beginning with "s".)

My main program uses a simple text-based interface to give the user access to the directory. (A GUI program might be preferable, but is much harder to write.) In a while loop, the program presents the user with a menu of options:

1. Look up a phone number

2. Add an entry to the directory

3. Delete an entry from the directory

4. Change someone's phone number

5. Exit form this program

The user is asked to select one of the options by typing in its number. For option 5, the program responds by breaking out of the while loop and exiting from the program. In the case of each of the other four cases, a subroutine is called to process the option. The subroutine gets the necessary data from the user and calls an instance method from the PhoneDirectory class to perform the requested operation. All this is fairly straightforward.

But the real point of the exercise is using a file to store the phone directory data. We need to decide on some format for storing the data in a file. One possibility -- and in some ways the easiest -- is to use ObjectOutputStreams and ObjectInputStreams to save and reload the directory. (These were discussed at the end of [Section 10.1](https://math.hws.edu/eck/cs124/javanotes3/c10/s1.html).) For this to work, the phone directory class would have to be declared to be serializable:

public class PhoneDirectory implements Serializable { ...

If this is done, then writing the directory a file named fileName is pretty simple. If directory is a variable that refers to the PhoneDirectory object, we could just say:

try {

ObjectOutputStream out; // For writing the object to the file.

out = new ObjectOutputStream( new FileOutputStream(fileName) );

out.writeObject(directory);

}

catch (IOException e) {

System.out.println("Whoops... Your data couldn't be saved.");

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

}

Reading the data from a file is almost as easy. It does require a type-cast to cast the Object returned by the readObject() method into a PhoneDirectory object. (This type-cast is one possible source of error, if the file contains a valid object but not an object of type PhoneDirectory.)

try {

ObjectInputStream in; // For reading the object from the file.

in = new ObjectInputStream( new FileInputStream(fileName) );

Object obj = in.readObject(); // Get an object from the file.

directory = (PhoneDirectory)obj; // Type-cast it.

}

catch (IOException e) {

System.out.println("Couldn't read directory data from file.");

System.out.println("Can't continue.... program aborted.);

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

System.exit(1);

}

I didn't take this approach in my program. One possible objection is that it produces a file in machine-readable form instead of human-readable. It's nice to be able to read the data in the file directly. It also ties the file very tightly to the PhoneDirectory class. It might be nice to be able to use the same data in other programs that use different classes to store the directory data. Instead, I designed a text-file format. The first line of the data file specifies the number of entries in the phone directory. After that, there are two lines for each entry. The first line is the name and the second is the associated number.

My PhoneDirectory class contains two instance methods for reading and saving data. (It's not clear that the PhoneDirectory class is the best place for these methods. Maybe the saving and reading should be done entirely in the main program. However, my PhoneDirectory class provides no way for the main program to get a list of all entries from the directory, so there is no way for the main program to save that list in a file. Arguably, it might have been preferable to add such a method to the PhoneDirectory class.) Here are the methods. The essential point is that the load() method must be able to read data in the exact format that is written by the save() method.

public void load(TextReader in) throws IOException {

// Clears any entries currently in the directory, and loads

// a new set of directory entries from the TextReader. The

// data must consist of the following: a line containing the

// number of entries in the directory; two lines for each

// entry, with the name on the first line and the associated

// number on the second line. Note that this method might

// throw an IllegalArgumentException if the data in the file

// is not valid -- for example if the same name occurs twice.

// Note that if an error does occur, then the original

// data in the directory remains.

int newCount = in.getlnInt();

String[] newNames = new String[newCount + 5];

String[] newNumbers = new String[newCount + 5];

for (int i = 0; i < newCount; i++) {

newNames[i] = in.getln();

newNumbers[i] = in.getln();

}

count = newCount;

names = newNames;

numbers = newNumbers;

changed = false;

}

public void save(PrintWriter out) {

// Saves all the entries in the directory to the PrintWriter.

// Data is written in the same format that is used in the load()

// method. Note that PrintWriters do not throw exceptions.

// To test whether the data was written successfully, the

// caller of this routine can call out.checkError().

out.println(count);

for (int i = 0; i < count; i++) {

out.println(names[i]);

out.println(numbers[i]);

}

}

Using these methods, the following code could be used to load a phone directory from a file named fileName:

try {

TextReader in = new TextReader( new FileReader(fileName) );

directory = new PhoneDirectory(); // A new, empty directory.

directory.load(in); // Try to load it with data from the file.

}

catch (Exception e) {

TextIO.putln("An error occurred while read data from \""

+ fileName + "\":");

TextIO.putln(e.toString());

TextIO.putln("Program aborted.");

System.exit(0);

}

In my program, the data is stored in a file named "phone.dat", but the user has the option of specifying a different file name as a command line argument when the program is run. If the file does not exist, the program offers to create a new, empty phone directory. Otherwise, it tries to read the phone directory data from the file. this makes the procedure for creating the directory somewhat more complicated. Here is the code that I actually use in my program to load the phone directory:

TextReader in; // A stream for reading the data.

try {

// Try to create a stream for reading from the file.

// If the file is not found, set the value of in to null.

in = new TextReader( new FileReader(fileName) );

}

catch (Exception e) {

in = null;

}

if (in == null) {

// The specified file could not be opened. Give the

// user the option of creating a new, empty file.

TextIO.putln("\nThe file \"" + fileName + "\" does not exist.");

TextIO.put("Do you want to create the file? ");

boolean create = TextIO.getlnBoolean();

if (create == false) {

TextIO.putln("Program aborted.");

System.exit(0);

}

directory = new PhoneDirectory(); // A new, empty phone directory.

try {

// Try to create the file.

PrintWriter out = new PrintWriter( new FileWriter(fileName) );

directory.save(out);

if (out.checkError())

throw new Exception();

TextIO.putln("Empty directory created.");

}

catch (Exception e) {

TextIO.putln("Can't create file.");

TextIO.putln("Program aborted.");

System.exit(0);

}

}

else {

// The input stream was created successfully. Get the data.

try {

directory = new PhoneDirectory(); // A new, empty directory.

directory.load(in); // Try to load it with data from the file.

}

catch (Exception e) {

TextIO.putln("An error occurred while read data from \""

+ fileName + "\":");

TextIO.putln(e.toString());

TextIO.putln("Program aborted.");

System.exit(0);

}

}

**The Solution**

**The PhoneDirectory class:**

/\*

An object of type PhoneDirectory holds a list of names and associated

phone numbers. In this simple implementation, both the names and the

numbers are stored as strings. The names and numbers must be non-null

strings, but no attempt is made to ensure that the values make sense.

Comparison of names is in all cases case-insensitive. A given name

cannot occur more than once in the directory. The instance methods

throw IllegalArgumentExceptions when the rules are violated.

The instance methods load() and save() are provided for loading

the data for the directory from a stream and for saving the data

to a stream.

\*/

import java.io.\*;

public class PhoneDirectory {

/\* The data for the directory is stored in a pair of arrays. The phone

number associated with the name names[i] is numbers[i]. These

arrays will grow, as necessary, to accommodate as many entries as

are added to the directory. The variable count keeps track of

the number of entires in the directory. \*/

private String[] names = new String[1];

private String[] numbers = new String[1];

private int count = 0;

public boolean changed; // This variable is set to true whenever a change

// is made to the data in this directory. The value

// is false when the object is created. The only time

// that it is reset to false is if the load() method

// is used to load a phone directory from a stream.

// (Programs that use the directory can also set the

// value of changed if they want, since it's public.)

public void load(TextReader in) throws IOException {

// Clears any entries currently in the directory, and loads

// a new set of directory entries from the TextReader. The

// data must consist of the following: a line containing the

// number of entries in the directory; two lines for each

// entry, with the name on the first line and the associated

// number on the second line. Note that this method might

// throw an IllegalArgumentException if the data in the file

// is not valid -- for example if the same name occurs twice.

// Note that if an error does occur, then the original

// data in the directory remains.

int newCount = in.getlnInt();

String[] newNames = new String[newCount + 5];

String[] newNumbers = new String[newCount + 5];

for (int i = 0; i < newCount; i++) {

newNames[i] = in.getln();

newNumbers[i] = in.getln();

}

count = newCount;

names = newNames;

numbers = newNumbers;

changed = false;

}

public void save(PrintWriter out) {

// Saves all the entries in the directory to the PrintWriter.

// Data is written in the same format that is used in the load()

// method. Note that PrintWriters do not throw exceptions.

// To test whether the data was written successfully, the

// caller of this routine can call out.checkError().

out.println(count);

for (int i = 0; i < count; i++) {

out.println(names[i]);

out.println(numbers[i]);

}

}

public String numberFor(String name) {

// Get the phone number associated with the given name, if any.

// If the name does not exist in the directory, null is returned. The

// name cannot be null. (If it is, an IllegalArgumentException is thrown.)

if (name == null)

throw new IllegalArgumentException("Name cannot be null in numberFor(name)");

int position = indexOf(name);

if (position == -1)

return null;

else

return numbers[position];

}

public void addNewEntry(String name, String number) {

// Create a new entry in the directory for the given name and number.

// An IllegalArgumentException is thrown if the name is already in

// the directory or if either of the parameters is null. If the

// arrays, "names" and "numbers", that hold the data are full,

// replace them with larger arrays.

if (name == null)

throw new IllegalArgumentException("Name cannot be null in addNewEntry(name,number)");

if (number == null)

throw new IllegalArgumentException("Number cannot be null in addNewEntry(name,number)");

int position = indexOf(name);

if (position != -1)

throw new IllegalArgumentException("Name already exists in addNewEntry(name,number).");

if (count == names.length) {

String[] tempNames = new String[ 2\*count ];

String[] tempNumbers = new String[ 2\* count];

System.arraycopy(names, 0, tempNames, 0, count);

System.arraycopy(numbers, 0, tempNumbers, 0, count);

names = tempNames;

numbers = tempNumbers;

}

names[count] = name;

numbers[count] = number;

count++;

changed = true;

}

public void deleteEntry(String name) {

// Remove the entry for the specified name from the directory, if

// there is one. If the specified name does not exist in the

// directory, nothing is done and no error occurs.

if (name == null)

return;

int position = indexOf(name);

if (position == -1)

return;

names[position] = names[count-1];

numbers[position] = numbers[count-1];

count--;

changed = true;

}

public void updateEntry(String name, String number) {

// Change the number associated with the given name. An IllegalArgumentException

// is thrown if the name does not exist in the directory or if either of

// the parameters is null.

if (name == null)

throw new IllegalArgumentException("Name cannot be null in updateEntry(name,number)");

if (number == null)

throw new IllegalArgumentException("Number cannot be null in updateEntry(name,number)");

int position = indexOf(name);

if (position == -1)

throw new IllegalArgumentException("Name not found in updateEntry(name,number).");

numbers[position] = number;

changed = true;

}

private int indexOf(String name) {

// Finds and returns the position of the name in the names array,

// ignoring case. Returns -1 if the name does not occur in the

// array.

for (int i = 0 ; i < count; i++) {

if (names[i].equalsIgnoreCase(name))

return i;

}

return -1;

}

} // end class PhoneDirectory

**The main program:**

/\*

This program serves as an interface to a (very simplistic) phone

directory. The directory is implemented as an object belonging

to the class PhoneDirectory. This object keeps a list of names

and associated numbers. A name can occur at most once in the

directory. When the program is loaded, the data for the directory

is loaded from a file. The name of the file can be given on

the command line; otherwise, it is given by the constant

DEFAULT\_FILENAME. If the data is changed while the program is

running, then the file is rewritten with the changed data before

the program terminates.

If no file with the given name exists when the program is

run, the user is given the option of creating a new, empty

phone directory file.

The user can perform a sequence of operations on the directory

chosen from this list: Look up a number, add an entry, delete an

entry, or modify an entry. This continues until the user chooses

to exit from the program.

\*/

import java.io.\*;

public class Phones {

static final String DEFAULT\_FILENAME = "phones.dat";

static PhoneDirectory directory; // Holds the data for

// the phone directory.

public static void main(String[] args) {

String fileName; // Name of file that stores the directory data.

boolean done; // Set to true when the user wants to exit the program.

/\* Get the file name from the command line, or use the

DEFAULT\_FILENAME if there is no command-line argument. \*/

if (args.length == 0)

fileName = DEFAULT\_FILENAME;

else

fileName = args[0];

/\* Read the phone directory data from the file. This routine

might terminate the program if an error occurs when the

attempt is made to end the data. \*/

readPhoneData(fileName);

/\* Show user a menu of available operations, get the user's

choice, and carry it out. Repeat until the user selects

the "Exit from this program" operation. Each of the other

four commands is carried out by calling a subroutine. \*/

done = false;

while (done == false) {

TextIO.putln();

TextIO.putln();

TextIO.putln("Select the operation you want to perform:");

TextIO.putln();

TextIO.putln(" 1. Look up a phone number");

TextIO.putln(" 2. Add an entry to the directory");

TextIO.putln(" 3. Delete an entry from the directory");

TextIO.putln(" 4. Change someone's phone number");

TextIO.putln(" 5. Exit form this program.");

TextIO.putln();

TextIO.put("Enter the number of your choice: ");

int menuOption = TextIO.getlnInt();

switch (menuOption) {

case 1:

doLookup();

break;

case 2:

doAddEntry();

break;

case 3:

doDeleteEntry();

break;

case 4:

doModifyEntry();

break;

case 5:

done = true;

break;

default:

System.out.println("Illegal choice! Please try again.");

} // end switch

} // end while

/\* If the phone directory data has been modified, write the

changed data back to the file. \*/

if (directory.changed == true)

writePhoneData(fileName);

TextIO.putln("\nExiting program.");

} // end main()

static void readPhoneData(String fileName) {

// Get the data for the phone directory from the specified

// file. Terminate the program if an error occurs. If the

// file does not exist, give the user the option of creating

// it.

TextReader in; // A stream for reading the data.

try {

// Try to create a stream for reading from the file.

// If the file is not found, set the value of in to null.

in = new TextReader( new FileReader(fileName) );

}

catch (Exception e) {

in = null;

}

if (in == null) {

// The specified file could not be opened. Give the

// user the option of creating a new, empty file.

TextIO.putln("\nThe file \"" + fileName + "\" does not exist.");

TextIO.put("Do you want to create the file? ");

boolean create = TextIO.getlnBoolean();

if (create == false) {

TextIO.putln("Program aborted.");

System.exit(0);

}

directory = new PhoneDirectory(); // A new, empty phone directory.

try {

// Try to create the file.

PrintWriter out = new PrintWriter( new FileWriter(fileName) );

directory.save(out);

if (out.checkError())

throw new Exception();

TextIO.putln("Empty directory created.");

}

catch (Exception e) {

TextIO.putln("Can't create file.");

TextIO.putln("Program aborted.");

System.exit(0);

}

}

else {

// The input stream was created successfully. Get the data.

try {

directory = new PhoneDirectory(); // A new, empty directory.

directory.load(in); // Try to load it with data from the file.

}

catch (Exception e) {

TextIO.putln("An error occurred while read data from \"" + fileName + "\":");

TextIO.putln(e.toString());

TextIO.putln("Program aborted.");

System.exit(0);

}

}

} // end readPhoneData()

static void writePhoneData(String fileName) {

// Save the data from the phone directory to the specified file.

PrintWriter out;

try {

out = new PrintWriter( new FileWriter(fileName) );

}

catch (Exception e) {

TextIO.putln("\nCan't open file for output!");

TextIO.putln("Changes have not been saved.");

return;

}

directory.save(out);

if (out.checkError()) {

TextIO.putln("Some error occurred while saving data to a file.");

TextIO.putln("Sorry, but your phone directory might be ruined");

}

}

static void doLookup() {

// Carry out the "Look up a phone number" command. Get

// a name from the user, then find and print the associated

// number if any.

TextIO.putln("\nLook up the name: ");

String name = TextIO.getln();

String number = directory.numberFor(name);

if (number == null)

TextIO.putln("\nNo such name in the directory.");

else

TextIO.putln("\nThe number for " + name + " is " + number);

}

static void doAddEntry() {

// Carry out the "Add an entry to the directory" command.

// This will only work if the name that the user specifies

// does not already exist in the directory. If it does,

// print an error message and exit. Otherwise, get the

// number for that person from the user and add the entry

// to the directory.

TextIO.putln("\nAdd entry for this name: ");

String name = TextIO.getln();

if (directory.numberFor(name) != null) {

TextIO.putln("That name is already in the directory.");

TextIO.putln("Use command number 4 to change the entry for " + name);

return;

}

TextIO.putln("What is the number for " + name + "? ");

String number = TextIO.getln();

directory.addNewEntry(name,number);

}

static void doDeleteEntry() {

// Carry out the "Delete an entry from the directory" command.

// Get the name to be deleted from the user and delete it.

// If the name doesn't exist in the directory, print a message.

TextIO.putln("\nDelete the entry for this name: ");

String name = TextIO.getln();

if (directory.numberFor(name) == null)

TextIO.putln("There is not entry for " + name);

else {

directory.deleteEntry(name);

TextIO.putln("Entry deleted.");

}

}

static void doModifyEntry() {

// Carry out the "Change someone's phone number" command.

// Get the name from the user. If the name does not exist

// in the directory, print a message and exit. Otherwise,

// get the new number for that person and make the change.

TextIO.putln("\nChange the number for this name: ");

String name = TextIO.getln();

if (directory.numberFor(name) == null) {

TextIO.putln("That name is not in the directory.");

TextIO.putln("Use command number 2 to add an entry for " + name);

return;

}

TextIO.putln("What is the new number for " + name + "? ");

String number = TextIO.getln();

directory.updateEntry(name,number);

}

} // end class Phones