Programming in Java

Fundamentals of Programming in Java

Assignment 1

Corrupted Database

Marks available: 100

Date issued: 08/10/2018

Deadline: 1700, 29/10/2018

Submission instructions

Submit your work by the deadline above. The accepted submission formats are a zip archive or a jar archive. Submit only .java source files in your archive. Do NOT submit .class files. Please do not submit Eclipse (or other IDE) projects.

Please make sure that the following package declaration is the first line of your Java file:

```
package com.bham.pij.assignments.a1a;
```

Do not output anything to the console unless explicitly asked to. You may, of course, use console output to debug your programs. Just make sure that you comment out or delete such debug statements when you submit.

Where applicable, your class for each of the exercises must have the name shown in the exercise heading.

In each of these exercises, you are not restricted to only creating the *required* methods. If you want to, you can create others. However, you must create **at least** the required methods.

Do not use any input method (such as `Scanner`) in any method that is required by this specification.

You must not use *regular expressions* in your solution. You are expected to confirm the validity or otherwise of each potential email address using `String` and `Character` operations that do not involve regular expressions. Use of regular expressions in your solution will result in a mark of zero.

Introduction

Your organisation has asked you to perform an important task. A database system that stores customer details has become badly corrupted. The result is that, as things stand, you have lost the email addresses of all of your customers and that was the primary method that was used to communicate with them. All that remains of the database table that contained the email addresses is a text file called 'corrupteddb'. If you open this file you will that, whilst it is readable to some extent, it now has no structure. There is no way that the database management system will be able to read this file. Your task is to write an application that will extract as many email addresses as possible from the file.

The organisation tells you that all of its customers resided in the USA, UK, Germany, Romania or Japan.

The email addresses

Email addresses have two parts to them, separated by the '@' symbol: the *local-part* and the *domain-part*.

In the following examples, the string before the '@' symbol is the local-part and the string after the '@' is the domain-part:

sole_survivor@sanctuaryhills.com
axlrose6@gunsnroses.net
gumball@amazing.world.of.jp

Some local-parts will have a *separator*. In the above examples, 'sole_survivor' has an underscore character '_' as a separator.

In the following description, the local-part and the domain-part will be described as having *components*. The local-part can have more than one component, separated by a separator. The domain part will have at least two components, separated by a separator.

The final part of the domain-part, which follows the final period ('.') character is called the *top-level domain*.

However, to simplify your task, you have been told that you only need to attempt to find email addresses that conform to the following format:

Local-part

The local-part will contain only:

- Upper-case and lower-case letters.
- The digits 0-9.
- The underscore character '_'. This can appear anywhere in the local-part.
- The period '.' character, with the restriction that this is only used as a separator. The period character cannot appear at the beginning or end of the local-part. There will be a maximum of two components to the local-part. For example, 'gumball' has one component whilst 'bill.gates' has two. There will be, therefore, at most one period character in the local-part.

Domain-part

The domain-part will contain only:

- Lower-case letters.
- The period '.' character, with the restriction that this is only used as a separator. Each domain-part will consist of either two or three components separated by a period.

You are also allowed to make the following assumption:

- Due to the information given about about the residencies of the customers, all domain-parts will top-level domain that is from the following set: {net, com, uk, de, jp, ro}. No other top-level domain is valid.

When your program is looking for email addresses in the file, it does not need to try and 'rescue' email addresses by correcting them. You do not have enough information to do that. Although some email addresses will look obviously wrong to you, your program should still find those email addresses. Your program may well find 'email addresses' that do not even exist (see next section) but are simply artefacts of the corrupted file. That is OK. However, see the remarks below about *embedded addresses*.

Your only task is to find strings that conform to the rules set out above. You are not expected to make a judgement about whether some email address is real or not.

Things to consider

Your program will be expected to work on other files. If another section of the database file was rescued at a later point, for example. This simply means that you should make sure your program works on any email address that conforms to the format above, **not** just the ones in the 'corrupteddb' file.

If a valid email address is embedded in an invalid one then you should extract the valid part. This means that, in a string of characters, if there is a sequence that corresponds to a valid email address, you should extract that. Do not discard the whole string simply because the string as a whole is invalid.

As an example, the local-part of an email address might have some invalid characters in it. If the characters *after* the invalid ones but before the '@' symbol are valid then treat *those* as a valid local-part.

For example, in the email address:

johnsmi$th@gmail.com

There is a valid email address: th@gmail.com

In this case your program should extract that valid email address.

However, consider the following address:

johnsmith∗@gmail.com

It is not possible to make a valid email address from this (in this application). There is no *consecutive* group of characters that represents a valid email address in this example. You might think to simply

exclude the * character but don't do that. Remember, you are not being asked to 'rescue' email addresses

As another example, consider this email address:

CUSTOMERPRIMARY_KEY6.phillip@facebook.com

This is a valid email address. You might be forgiven for suspecting that the *real* address is 'phillip@facebook.com' but you are not justified in doing so. In this case you would return the full, valid email address above.

False positives and false negatives

A *false positive* is the erroneous identification of an instance that is not actually there. In the case of the corrupted database, a false positive would result from some section of the file that seems to represent a real email address but, in fact, is not anybody's email address. However, it is impossible for you to know the difference (somebody *might* have the name 'CUSTOMERPRIMARY_KEY6' ...), so don't try. Your task is simply to find any strings in the file that conform to the above email format. Whether they are actually all *actual* email addresses or not is a different matter that you are not addressing. You cannot avoid false positives, provided you have made sure that all of your identified email addresses have the valid format.

A *false negative* is the erroneous rejection of a valid instance. In the case of the corrupted database, a false negative would occur if your application rejected a string that actually consisted of a valid email address, whether it is anybody's actual email address or not. You should try and avoid false negatives.

Implementation

You need to download the file `EmailAddressFinder.java`. This class contains a program. You will write one method in this program that has the following signature:

```
public ArrayList<String> findEmailAddresses(String input)
```

Do not vary this signature (You may change the name of the parameter, but not the type).

Your `findEmailAddresses()` method will be called (possibly repeatedly) by the provided program with sections of the text from the corrupted database file passed as the parameter `input`. The program will scan the file and then pass sections of it to your method based on where it finds newline characters in the file. It might be the case that, in fact, it passes you the entire file as a single codeString.

You don't need to concern yourself with that part of the process, however. You simply need to implement this one method. This method, as implemented by you, will search the input string for email addresses. The string that is passed to your method might have zero or more email addresses in it. Any email addresses your method finds should be added to an `ArrayList` and then the `ArrayList` should be returned by the method. This method must return an `ArrayList` of `String`s even if no email addresses were found. In the latter case, the list should exist but be empty.

Hints

You need to consider how you will identify the boundaries between sequences of characters that represent valid email addresses and those that don't. One of the main ways you can do this is to look for strings with spaces either side of them since no valid email address can contain a space. Also, in order to help you, your task has been limited to locating only email addresses with the top-level domains listed above.

You might consider how you will identify where the top-level domains are in the input string.

Some cases will be easier than others. If an email address happens to have space characters either side of it, the input string will contain only that email address. Your task will then simply be identifying whether that email address is valid or not. Your method may, however, be passed a larger section of text that contains any number of email address or none. Within this section of text, there may be addresses that are not separated from each other by any other character.

You should initially concentrate on dealing with the more straightforward case in which your method receives a single email address. This will help you develop your algorithm for detecting valid email addresses.

Spend some time looking at the 'corrupteddb' file in a text editor. Identify cases that you think look easiest to deal with and implement what you need to identify those first before moving on to the less straightforward cases.