Problem Description

Mining information from sales data is a common task in both traditional and online stores. Past sales information can guide purchasing and enable the company to make recommendations to customers about what they may like to buy.

This assignment performs two data mining tasks. First, it determines the ranking of products according to how many of each product has been sold. Second, it also determines the five other products most commonly bought with each product.

While these tasks are not particularly complex, performing these tasks on large data sets is computationally expensive and the use of data structures and algorithms that facilitate fast searching and sorting is desirable.

Assignment Requirements

Write a Java program called `SalesInfoMiner` that reads in a file of products, stored in text format (2 lines per entry). The program must then read in information about past sales transactions and output all the products to a new file, sorted by product description, along with their sales rank and the five other products with which they were most commonly purchased.

The program accepts four command line arguments and *must not expect any user interaction from the keyboard*. The first parameter to the program is the name of the product list to read in; the second parameter is the file containing information about past transactions. The third parameter is the name of the file to which the ordered product list will be written, and the fourth parameter is the name of the file to which any error or warning messages will be output.

Product File

This file contains data for one or more products

- Each product occupies two lines
- Line 1 contains the product id, which is a *string* of 13 digits
- Product IDs are unique
- Line 2 is the description of the product, which is not an empty string
- You can assume that the file is not empty.

Thus, the data for a product has the format shown below:

`<ProductID>` (A thirteen digit string)
`<Product Description>` (A text string)

For example:

```
1234567890123
Milk – full cream 600ml
9832983102938
Bread – wholemeal loaf
1299128391283
Eggs, Extra Large, Barn laid
```

Sales Transaction File

This file contains data for one or more sale transactions

- Each sale transaction has *2 + n* lines, where *n* is the number of items in this sale transaction (as specified on line 2 of the sale transaction)
- Line 1 has the format `Receipt number: <receipt number>` where `<receipt number>` is a positive integer
- Receipt numbers are unique
- Line 2 has the format `Number of items: <number of items>` where `<number of items>` is a positive integer
- Each sale item occupies a line, which contains the product id of the item
- All product IDs in a sale transaction are *distinct* and must exist in the product file
- You can assume that the file is not empty

Thus, data for a sale has the format shown below:

```
Receipt number: <ReceiptNumber>     (<ReceiptNumber> is an integer)
Number of items: <NumberItems>      (<NumberItems> is an integer)
<ProductID>
...
```

For example:

```
Receipt number: 1
Number of items: 3
1234567890123
9832983102938
1299128391283
Receipt number: 2
Number of items: 4
9832983102938
1234567890123
1299128391283
1234567890123
```

Product Output File

- For each product, the program determines its sale transaction count, i.e. number of transactions that the product appears in.
- The program lists the products in an output file, sorted by sale transaction count, in decreasing order.
- When there is a tie in the sale transaction count, the products are listed in alphabetical order of their descriptions (see example below)
- For each product, the program also lists the top 5 products that are most frequently bought with this product.
- The "top 5" list may have less than 5 products. It can also have for than 5 due to ties. Whenever there is a tie in the frequency, the products are listed in alphabetical order of their descriptions (see example below)
- The format is as shown below

```
Product: <Product Description>
ID: <ProductID>
Sales Count: <SalesCount>
This product was purchased most often with:
1: (<ProductID>) - <Product Description>
2: (<ProductID>) - <Product Description>
3: (<ProductID>) - <Product Description>
4: (<ProductID>) - <Product Description>
5: (<ProductID>) - <Product Description>
```

For example:

```
Product: Milk – full cream 600ml
ID: 1234567890123
Sales Count: 82
This product was purchased most often with:
1: (1299128391283) Eggs, Extra Large, Barn laid [8]
2: (9832983102938) Bread – wholemeal loaf [5]
3: (3289742398473) Apples – Golden Delicious [4]
4: (9348029348934) Chocolate, Dark, 200g [4]
5: (3024823094839) Noodles, Udon – Fresh 250g [4]
6:   (0002011106917) Nuts, almonds, dry roasted [4]
Product: Pork - cured, ham, separable fat, boneless
ID: 0000140956153
Sales Count: 82
This product was purchased most often with:
. . .
```

"Sales count" is the number of transactions that a product appears in. The numbers in square brackets are the number of transactions a product appears with another product

Calculating the most frequent items purchased with a product: With every product, we need to output the 5 other items with which it was most frequently purchased. These other items must be listed in order of frequency. The item listed at position 1 would have been bought more frequently with the item than the item at position 2, and so on. Though we are interested in the "top 5", the list may have more than five items due to ties. In that case, more than five items are to be included and items involved in a tie are to be ordered by product description. If a product has not been bought with 5 other items, then less than 5 items are output.

Error File

In the sales transaction file, a sales transaction "record" is considered to include all the lines from the line starting with "Receipt number"

- Up to, but not included, the next line starting with "Receipt number:"
- Or up to the end of the file.

When a particular line does not conform, as expected, to the description given above,

- the error should be detected
- an error message is written to the error file (the fourth command-line argument), which indicate
    - the type of the error
    - the file in which the error occurs
    - and the line number of the line on which the error occurs
- no result ~~(i.e. listing of products as described in Section B)~~ is output**, i.e. the product output file (the third command-line-argument) is empty.**
- and the **program terminates**

~~Based on the specification in Section A,~~ **Based on the descriptions of the product file (the first command-line-argument) and the transaction file (the second command-line-argument)**, we can have the following types of data error (in addition to errors such as file does not exists, etc.)

For product file:

1. Product ID is invalid – not consisting of 13 digits
2. Product description is empty/missing
3. Product ID is not unique

For sales transaction file:

1. Line 1 of a sale transaction does not start with "Receipt number:"
2. Receipt number (on line 1) is not a positive integer
3. Line 2 of a sale transaction does not start with "Number of items:"
4. Number of items (on line 2) is not a positive integer
5. Product ID does not consists of 13 digits
6. Product ID in a sale transaction has duplicates
7. Product ID does not exist in the product file
8. There are not enough product ID's listed for a particular transaction (i.e. less than the number listed on line 2)

Note: If there are more product ID's listed than expected, we could detect it with the error of type 1 above.

The error messages will have the following format:

```
ERROR: <Error Name>
<Error Description>
<Solution>
```

For example:

```
ERROR: Invalid Product ID
Product 8763726273678 in receipt 4 is not in the product file
Product discarded, processing of transaction continued
The program is terminated
```

You can assume that in the sales transaction file

- There are no blank lines
- There are no duplicated receipt numbers

Restriction on the Use of Classes in the Java Collection Framework

Except for ArrayList and Linkedlist, you are not permitted to use any other classes in the Java Collection Framework.

Testing Your Program

You will test your program against some large data sets, which will be posted on LMS. When testing your program against large data set, please do not run your program on the *latcs6* server, use the *simula* server instead. For some large data sets you may need to request, especially when you test your program on Windows, more memory by using the `-Xmx` option of the `java` command.

Execution of the Program

The program will be run with the following command:

```
java SalesInfoMiner <products> <transactions> <output-file>
<error-log>
```

If an incorrect number of arguments is supplied, then the program must output an error message explaining the arguments required before terminating.

Important Notes:

1. ***You must <u>not</u> submit your programs in a zip file, or similarly compressed file.***
2. ***Your classes must <u>not</u> belong to a package (i.e. having the package statement at the beginning of the class.***
3. ***If your program does not compile, you must write so in the report.***

Failing to comply with any of these requirements will result in <u>10 marks</u> being <u>deducted</u> from your assignment.

Marking

You are required to submit electronically a *written report* describing your solution and implementation, as well as your program code. The report is worth 30 marks while the program is worth 70 marks.

Correctness and efficiency are the major consideration in the creation of this program. However, code style, modularity and documentation are also important. The program will be marked on style, correctness and efficiency. Style will be worth 10 marks. This will include commenting, layout, choice of identifier names, choice of methods and/or classes, etc. Correctness and efficiency will be allocated 30 marks. 30 marks will also be allocated for the program's efficiency. Programs that are highly inefficient may receive no marks here.

There are files that will be provided on which the program should be tested. These are valid files and, at a minimum, your program must process these. You should also create data files, with and without errors, in order to test you program's error handling capabilities.

The report should describe:

- How the program works. You should focus on the data structures and algorithms used. Include in your discussion a comparison of at least three data structures/algorithms you considered using to solve the problem. (3 page limit) [10 marks]
- The tests carried out on the program, and the results of these tests. Testing should be carried out for both correctness of algorithm [10 marks] and efficiency [10 marks]
- Correctness testing should be reported in table form with fields for the test case's description, the test data used for this test, the expected outcome, and the actual outcome. At least 10 different test cases should be reported.
- Efficiency testing should include the results of using the time command on different data sets and a discussion of these results in relation to the theory of the data structures/algorithms used and considered. (3 page limit)

The report does not have to be long but it should cover all necessary issues. Point form and tables are acceptable where appropriate.

It should be submitted electronically as a file with the name "`report`". Acceptable formats are: plain text, PDF or Word. ***Make sure your name and student number are on every page of your report.***

■