Practice Assessment July 2022

Tags	
Complete	☐

Add a `.gitignore` file to ignore the `target` directory

Task 1 (30 Marks)

Mail merge is a feature within most data processing applications that enables users to generate documents from a single template with a data source that contains information like the recipient's name, address, etc. to be inserted into the template.

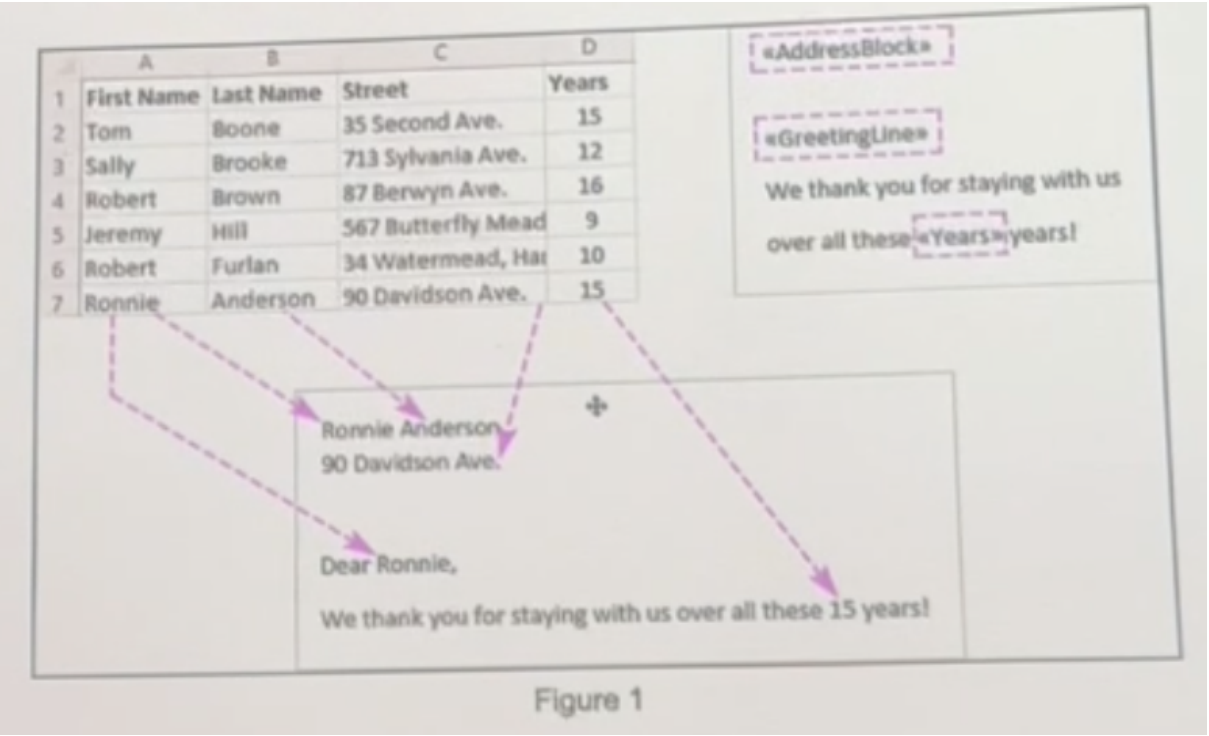

Figure 1

The above diagram illustrates how mail merge works. In Figure 1, the data from a spreadsheet (left) is inserted into a template (right) to produce the mail

(below centre). Each row from the data source will product a single document created by combing the row data with the template. The mail merge process in Figure 1 will produce 7 mails.

Use Maven to generate a Java project, call the project `task01`. You can use any package name. This should create a directory called `task01` in your repository.

Write a Java program to implement mail merge; your Java program takes 2 parameters from the command line:

- CSV (comma separated file) file containing all the data to be merge
- template file to be merged with the CSV data source

The mail merge program should be run in the following way

```
java your.java.Main <CSV file> <template file>
```

Data Source (CSV) File Format

Your Java mail merge should read a CSV file of the following format

- the first row consists of variable names (no space) to be substituted into the template)
- subsequent rows are the data

The following is an example of the CSV file

```
firs_name,last_name,address,years
Sherlock,Holmes,221b Baker st\nLondon,22
Harry,Potter,4 Privet Drive\nLittle Whinging,2
Homer,Simpson,742 Evergreen Terrace\nSpringfield, 10
```

The first line defines the variable name that should be bound to each row; for example if you are reading the third row (Harry), then the following values will be bound to the variables

Variable Name	Value
first_name	Harry
last_name	Potter
address	4 Privet Drive\nLittleWhinging
years	2

The values of the variable changes as your program reads each row of the CSV data source.

Template File Format

The template file consists of literal text and variables (defined in the corresponding CSV data source). Variables are prefixed and suffixed with a double underscore (__).

The following is an example of a template file that corresponds to the previous CSV file

```
__address__

Dear __first_name__,

Thank you for staying with us over these __years__.
```

When the template is applied to the third row (`Harry`) of the CSV file (example above), it produces the following text

```
4 Privet Drive
Little Whinging

Dear Harry,
```

```
Thank you for staying with us over these 2.
```

Your program should print out the ‘filled-in’ template after processing every line of the CSV file.

You are provided with 2 sets (template and corresponding CSV data file) of files to test your mail merge program. The invigilator will let you know where to download them.

Assume that there are no errors in the CSV and template files.

IMPORTANT: Your application must work with any CSV data source and template files that conforms to the above given file specific and not just the 2 sets of given samples.

Task 2 (25 Marks)

Use Maven to generate a Java project; call the project `task02`. You can have any package name. This should create a directory call `task02` in your repository.

Write a Java network client application that will exchange a series of messages with a server running at `sfd.chuklee.com` on port `80`.

The following diagram (Figure 2) illustrates the interaction between the client (which you will be writing) and the server.

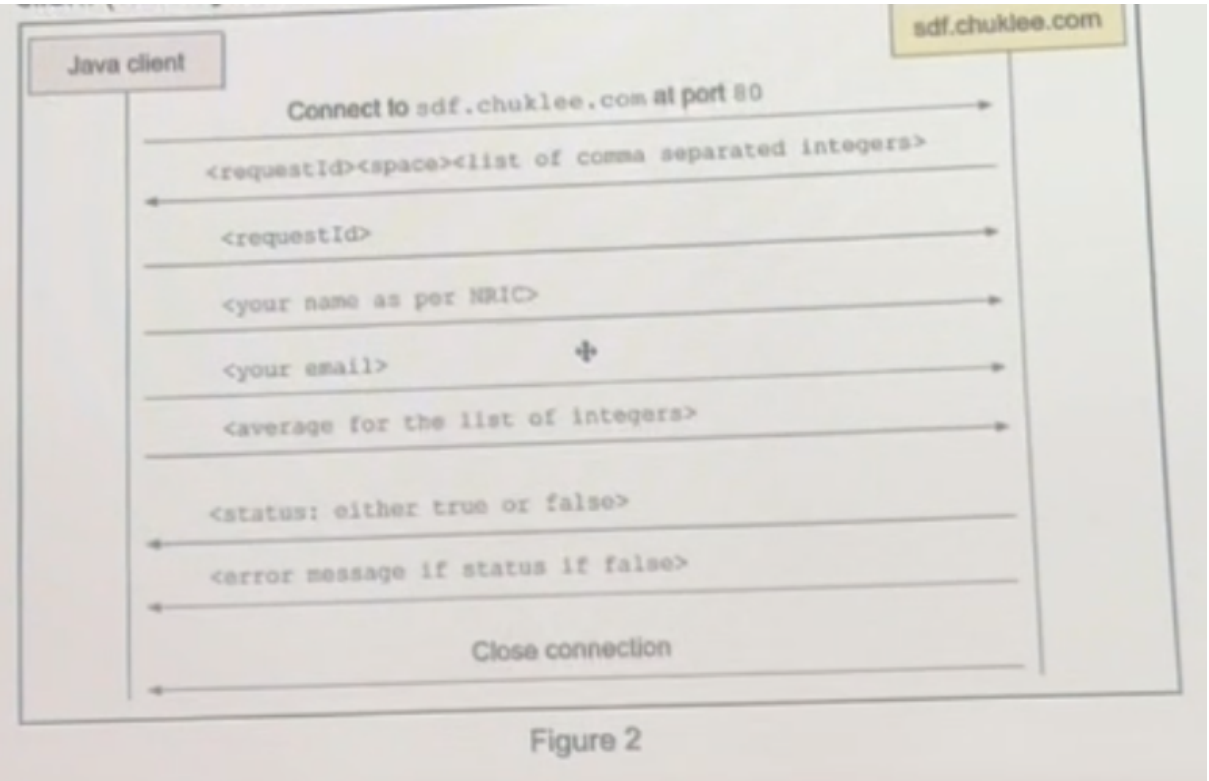

Figure 2

The following is a detailed description of Figure 2

1. The client (your Java application for this task) connects to the server at `sdf.chuklee.com` on port `80` .
    - When the connection has been established, get the `OutputStream` first followed by the `InputStream` from the socket. The order is important or your program will 'hang' if you try to get the `InputStream` first
    - Once you have got the `OutputStream` and `InputStream`, create the corresponding `ObjectOutputStream` and `ObjectInputStream` respectively

2. Use `readUTF()` to read the initial request from the server. The request is a string with 2 fields separated by a single space
    - request id - a unique hex string
    - comma separated list of integers
    - The following is an example of the request from the server:

```
1234abcd 97,35,82,2,45
```

- where `1234abcd` is the request id and `97,35,82,2,45` is the list of numbers

3. Calculate the average for the list of integers, eg. `55.2` for the example from the above list of numbers
    - Hint: in Java when you divide 2 integers, the answer will be an integer (eg. 5/2 = 2), but if either the numerator or the denominator is a float, then the answer will be a float (eg. 5/2.0 = 2.5)

4. Write the answer back to the server in the following sequence using methods from `ObjectOutputStream` listed below
    - The request id that you received from the server - writeUTF()
    - Your name as per NRIC - writeUTF()
    - Your valid email - writeUTF()
    - The average for the list of numbers - writeFloat()

5. After sending the above 4 items, the server will send either a true or a false to indicate the status of your response. Read the response from the server with the `readBoolean()` method.
    - If the response is true, close the socket connection. Print “SUCCESS” before the program exists
    - If the response is false, the server will send the reason for the failure. Read the error message with readUTF() and close the socket connection. Print “FAILED” followed by the error message.

The following is a flowchart of the interaction

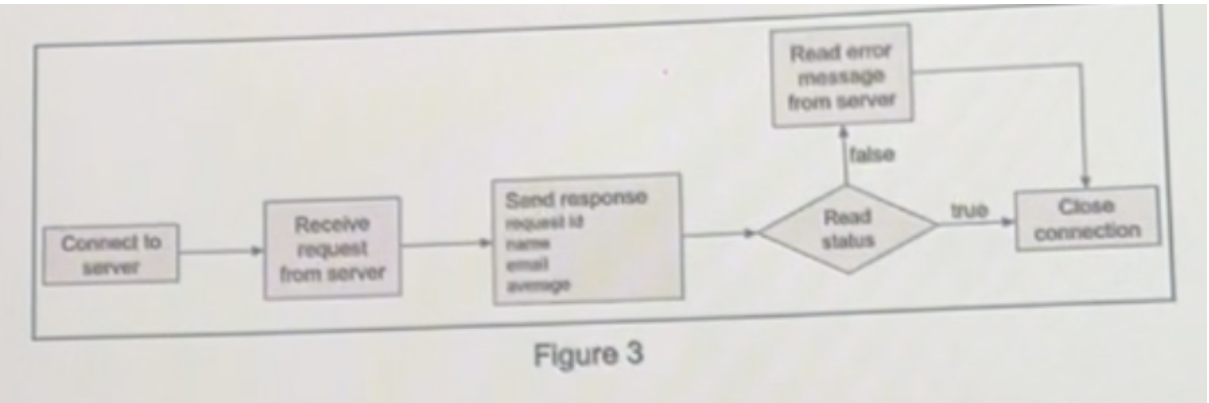

Figure 3

Once you have connected to the server, your program will have 15 seconds to complete the entire message exchange. If the duration exceeds 15 seconds, the server will close the connection