DALHOUSIE UNIVERSITY
Inspiring Minds

Final Project

SDC 3901

Author	Vamsi Krishna Utla
Email	vm271757@dal.ca
Student ID	B00870632
Created Date	25-11-2020

DALHOUSIE UNIVERSITY
Inspiring Minds

1.0 External Documentation

1.1 Overview

This program is used to replicate the scaled-down functionality of Canadian federal government's application for contact tracing of COVID-19. Through this program, it is possible to notify people (to self-quarantine) who have been in contact with other people who are COVID positive which will ultimately lead to limit the spread of disease at a faster pace. This program is also used to detect the frequency of large gathering which helps to understand the community's compliance with physical distancing advisories.

1.2 Files and external data

The implementation is carried out using seven significant files which are as follows:

- Contact.java – this is used to define the overall structure of a contact node.
- MobildeDevice.java – this file is used to register/create mobile device and store the details associated to its contacts and positive tests which are synced with government's server.
- Government.java – this class is used to read the contacts and test details from a mobile device and uploaded them to database along with providing the frequency of large gatherings.
- InvalidInputException.java – this file is used for implementing the custom exception that is used specifically to catch general exceptions.
- MainUI.java – this contains the main method for reading input and interacting with MobileDevice and Government classes for the purpose of COVID-19 contact tracing.
- JunitTestCases.java – this JUnit file contains the test cases required for testing the functionality of both MobileDevice and Government classes.
- raw_sql_statements.sql – this contains the database schema designed for this problem.

In addition to above, there are two separate configuration files that are used to create objects of MobileDevice and Government classes.

1.3 Data structures and their relations to each other

MobileDevice

This class mainly uses the following two data structures for storing contacts and positive test hashes:

- An array list of type 'Contact' that is used for storing the contact details of all the contacts of a mobile device. Each entry of the array list i.e., a contact in turn consist of details such as contact's hash, duration, date and a flag to identify if it has been synced with central database or not.
- An array list of type 'String' that is used for storing the positive test hashes reported by an agency.

Government

The following data structures are used for implementing the Government class:

- The mobileContact() method mainly uses the below two data structures for reading contacts and positive tests from a mobile device and writing the same to database.
    - An array list of type 'Contact' that is used for storing the contact details of all the contacts read from a mobile device.
    - An array list of type 'String' that is used for storing the positive test hashes that are also read from a mobile device.
- The findGatherings() method uses the below data structures to calculate the number of large gatherings on a given day.
    - A map of type 'String' as keys and 'boolean' as values which is used to store unique contacts that were recorded on the given day as keys and a default boolean value as false indicating the contact has not yet been considered as part of any large gathering.
    - A list of type 'String' which contains the same unique contacts from the above map. This is used to traverse the list of contacts in a sequential order for processing the gatherings.
    - A set of type 'String' used to store unique contacts that satisfy the condition that they have met pair of contacts i.e., 'A' and 'B' on the same given day. It is used in order to find a gathering.
    - A list of type 'String' used to store contacts from the above set so that they can be traversed in a sequential manner for finding large gatherings.

- A set of type 'String' used to store unique contacts that satisfy the condition that they have been in contact with other device for at least given amount of duration on the same day (as given in the method parameters).

1.4 Assumptions

The following are the list of assumptions for this problem:

- The positive test of a contact is considered for reporting the result in mobileContact() method only if it is also been recorded by an agency to the government server.
- The durations of multiple contacts with same device on the same day needs to be added.
- If the devices sync after 30 days of contact, then such contacts should not be considered while reporting in mobileContact() as they do not fall under 14 days period.
- If the contact got positive after contact date, still the mobileContact() should be returning true as the date of contact and test result date falls in 14 days period.
- If a mobile device gets in contact with a COVID positive contact, then they are reported as true via mobileContact() and synchronizeData() methods i.e., they have been in contact with a COVID positive person (within last 14 days). However, if the same positive contact gets in touch with the same mobile device again in the next 14 days, then they should be reported true again (if the test results still falls within 14 days) as the recent contact should be treated as a new one.
- If there is no data to sync, the mobileContact() should still be able to determine whether the initiator has been in contact with any COVID positive person in the last 14 days based on the information present in the database.
- When there are no contacts against the mobile device in the database, the methods mobileContact() and synchronizeData() always returns false.

1.5 Design elements

Database design

The database mainly consists of two tables which are used for recording contact details of a mobile device and test details of COVID-19 reported by an agency.

1. The first table consists of the following columns along with the specified details to store contact information:
    - contact_from
        - this column has a data type as varchar(100) and does not allow null values. It stores the device hash of the mobile device that syncs its contacts to the database.
    - contact_to
        - this column has a data type as varchar(100) and does not allow null values. It stores the device hash of the device that first column has been in contact with.
    - date
        - this column has a data type as int(11) and does not allow null values. It stores the date when devices in column one and column two have been in contact.
    - duration
        - this column has a data type as int(11) and does not allow null values. It stores the duration for which devices in column one and column two have been in contact with.
    - flag
        - this column has a data type as int(11) and does not allow null values. It stores '0' when the contact has not yet been considered as part of mobileContact() method of Government class and '1' when the contact has already been considered to report the device in column one that they have been in contact with a COVID positive (column two) in last 14 days.

    This table consists of a primary key with combination of column one, column two, column three and column four to uniquely identify the rows in the table.

2. The second table is used to store the test details and consists of the following attributes with specified details:
    - contact
        - this column has a data type as varchar(100). It stores the device hash of the device that this particular test belongs to.

- test_hash
  - this column has a data type as varchar(100) and does not allow null values. It stores the unique identifier of each test i.e., 'test hash'. It also acts as a primary key to this table.
- date
  - this column has a data type as int(11). It stores the date on which this test was taken.
- result
  - this column has a data type as tinyint(1)/boolean. It stores the result this test i.e., either '0' (false) or '1' (true).
- verification_status
  - this column has a data type as int(11) and does not allow null values. It stores '0' when the test is not yet verified or submitted by agency but has been reported by the mobile device and '1' if it has been recorded by the agency.

The detailed schema of the above is given in the following .sql file.

In addition, the following is the ER diagram representing the above two tables that are used as part of this problem implementation.

contact_details
*contact_from
*contact_to
*date
*duration
flag

test_results
contact
*test_hash
date
result
verification_status

Encoding format

The method synchronizeData() from MobileDevice sends the contacts and positive tests information to mobileContact() method of Government for syncing purpose. However, it is required to follow a proper format so that the data packing and unpacking becomes efficient during the process of synchronization.

This problem follows the XML format for packing the data associated to contacts and positive tests in MobileDevice into a string variable and passing it to mobileContact() of Government.

Sample output:

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<Contacts>
        <Contact>
                <ContacHash>1345687432</ContactHash>
                <Date>343</Date>
                <Duration>35</Duration>
        </Contact>
</Contacts>
<Tests>
        <Test>345679876</Test>
</Tests>
```

Please note that when there is no data to report to government server, the string variable contains only the version tag along with contacts and tests open and close tags. The sample output is as follows:

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<Contacts>
</Contacts>
<Tests>
</Tests>
```

Configuration files format

A configuration file is required for creating a mobile device and government object in this problem. However, there are some differences between the configuration files used for creating mobile device and government objects.

- The configuration file of a mobile device has two rows of data as follows:
    - address
    - deviceName
- The value and key are separated by “=” and the following is a template for mobile device’s configuration file:

- The configuration file of government has six rows of data as follows:
    - database
    - user
    - password

  - dbName
  - contactsTable
  - testsTable
- The value and key are separated by "<space>" and the following is a template for government's configuration file:

1.6 Key algorithms and implementation details

Contact

- This class is used to define the overall structure of a contact node that consists of the following attributes:
  - contactHash -> unique identifier of a device
  - date -> date of contact
  - duration -> duration of contact
  - sync -> flag to indicate if the contact has been synced with database or not
- It contains the following two constructors which are used to initialize the class members.
  - contactHash() -> used to set sync as false
  - contactHash(String contactHash, int date, int duration) -> used to set contact hash, date and duration along with sync as false.
- This class also contains the following getter and setter methods:
  - String getContactHash()
  - int getDate()
  - int getDuration()
  - boolean getSync()
  - setContactHash(String contactHash)
  - setDate(int date)
  - setDuration(int duration)
  - setSync(boolean sync)

MobileDevice

MobileDevice(String configurationFile, Government contactTracer)

- The constructor uses default hash function to develop a hash value with the help of address and device name which are given as part of configuration file. This hash value is used as a unique identifier for each mobile device.
- It uses an instance of BufferedReader to read the contents from the file on a line to line basis.
- The format of the configuration file is specified in section 1.5.
- In addition, the instance of contact tracer is also stored locally for later processing.
- The constructor throws InvalidInputException in case of any general issues such as invalid inputs, null government object etc., along with required details. It also throws IOException in case of any issues in accessing the configuration file.

recordContact(String individual, int date, int duration)

- This method creates a contact with given details and appends the contact to the array list of contacts.
- It is usually triggered when a mobile device comes in contact with another mobile device.
- The method also add any duplicate contacts as the duplication will be verified and avoided before they are synced with database in government class.
- By default, the flag for each new contact is set to false so as to represent that the contact is still not yet synced with central database of the government.
- The method throws InvalidInputException with a suitable message in case of any issue with the input parameters passed to this method.

positiveTest(String testHash)

- This method appends the new test hash that is passed as an input parameter to this method to the array list of positive tests.
- It is usually triggered by the testing agency to inform the user of the mobile device that the test they have given has come out as positive.
- The method checks for duplication scenario and ignores the testHash if it has already been added to array list of positive tests previously.
- It throws custom exception i.e., InvalidInputException in case of any issue with respect to the testHash passed as an input parameter to this method.

synchronizeData()

- This method formats the data in an XML format as per section 1.5 by following the below sequence of steps:
    - Initially, this method appends the header of the XML to the result string.
    - It will then append the 'Contacts' open tag on a new line to string.
    - The method will then iterate through the array list of contacts and appends them to the result string if the contact has not already been synced previously. The details such as contact's unique hash, date and duration is appended as a separate row with the help of predefined tags of 'ContactHash','Date' and 'Duration'.
    - The method will also make sure to add the closing tags so as provide a proper nesting to tags.
    - The method will then append the 'Tests' open tag on a new line indicating that the next section is associated to details of positive tests.
    - It iterates through the array list of positive test hashes that were recorded as part of 'positiveTest()' and appends the positive test hashes to the resultant string along with open and close tags of 'Test'.
    - In the end, the end tag of 'Tests' is appended to close the XML format without any issues.
    - Sample format of the XML:

- Finally, this method calls mobileContact() with the help of the contactTracer which is an instance of type Government by passing input parameters as the device's hash as initiator and the resultant string as contactInfo.
- The method will also return true/false to identify if the device has been in contact with other device whose user/owner has been tested positive for COVID within 14 days of their contact.
- In addition, the method also handles exceptions from mobileContact() and throws them to respective caller methods with necessary details. Some of the exceptions are as follows:
    - InvalidInputException in case of any general issues
    - IOException in case of any issues related to input/output format
    - ClassNotFoundException in case of driver manager issues
    - SQLException in case of any issues related to SQL queries
    - ParseException in case there is any issue while parsing a string.

Government

Government(String configurationFile)

- This acts as a constructor to Government class which is used for setting up the configurations required for connecting to the given database.
- The constructor uses Bufferedreader and FileReader for reading the content from the given configuration file on a line to line basis based on the format specified in section 1.5.
- It will then try to establish a connection with the given database to verify the given details.
- In case setting up a connection with the given database fails, the constructor will set the connection flag to false indicating that the given details with respect to database are invalid.
- As this flag is declared as a class member, it is used by other methods to identify the connection status and throw required exceptions accordingly as every method needs a proper database connection to carry out the tasks.
- It will throw the following exceptions with necessary details:
    - InvalidInputException in case of any issues with connection or format of the given configuration file.
    - IOException in case of any issue with reading content from the given configuration file.
    - ClassNotFoundException in case of any issues with JDBC driver manager.
    - SQLException in case of any issues with SQL statements.

boolean mobileContact(String initiator, String contactInfo)

- This method will initially check the connection flag that was set in the constructor and throws exception custom exception with required details.
- The method then checks for the version/encoding tag that should be present as the first non-empty line in the given string of contactInfo.
- It then moves on to calculate the current date as a number of days that is used in section three.
- The method also sets up connection with database that is used in sections two and three for reading and writing data to the database.
- This method is divided into three sections based on the implementation criteria:
    - Read the contacts and positive tests details.

- ❖ The method uses BufferedReader and StringReader for reading content from contactInfo on a line to line basis based on the format specified in section 1.5.
- ❖ Based on the XML format specified in section 1.5, different tags are predefined which are used to identify the kind of data existing in the current line and add them to either array list of contacts or array list of positive tests.
- ❖ The reader traverses until end of file and throws custom exception i.e., InvalidInputException in case of any issues with the format of the contactInfo.

- Write the data that is read in first step to the database.
  - ❖ In this section, initially contacts are read from the array list of contacts in a sequential manner. If a contact does not exist in the database, then it is inserted as a new record to contact's table of the database. Else, it will then check the date of the contact and insert as a new record in case the date is different or updates the duration (sums up duration) in case the date is same.
  - ❖ Secondly, the method will then try to traverse through the array list of positive tests and check if the test already exists in the database or not. If yes, then it will update the test row with initiator details (in column one i.e., contact) to indicate that the test belongs to the current initiator. Else, if the test is not present, then a new record of test is inserted with details of column one i.e., contact, test hash as column two and flag as zero in column five indicating that this test has not yet been verified by the agency.
- Identify if the initiator has been in contact with any other user who has been tested positive within 14 days of their contact.
  - ❖ This method will initially get all the contacts who have been in contact with initiator by satisfying the condition that they have not previously been considered already i.e., the flag of the contact in the table is not set to 1.
  - ❖ Identify which contacts have a test result as positive in the test's table of the database.
  - ❖ Based on the contacts obtained in previous step, the method checks whether these contacts have been made in the last 14 days or not.
  - ❖ If yes, then an additional check to verify if the difference between the contact date and test date is within 14 days or not. If the difference is

between 0 to 14 days, then the result is set as true and the contact flag is set to 1 so that it wouldn't be considered next time.

- ❖ Similarly, all such contacts are traversed and then the end result is returned to indicate the contact status after closing all the streams.

- This method throws the following exceptions along with required details:
  - InvalidInputException in case of any general issues
  - IOException in case of any issues related to input/output format
  - ClassNotFoundException in case of driver manager issues
  - SQLException in case of any issues related to SQL queries
  - ParseException in case there is any issue while parsing a string.

void recordTestResult(String testHash, int date, Boolean result)

- This method will initially check the connection flag that was set in the constructor and throws exception custom exception with required details.
- The method will then try to setup a connection to database (if connection exists) that is used to read, update and insert the data into test's table of the database.
- It will then identify if there are any records in the test's table with the same testhash. If yes, then it will update the details and set the flag to 1 indicating that the test record has been verified by the agency.
- In case there is no test record found with same testHash, then this method will insert a new record with the test details and leaves the column one as blank which will be updated when the contact syncs with the government.
- The method will then close all the stream to avoid any issues.
- This method throws the following exceptions along with required details:
  - InvalidInputException in case of any general issues
  - ClassNotFoundException in case of driver manager issues
  - SQLException in case of any issues related to SQL queries

int findGatherings(int date, int minSize, int minTime, float density)

- This method will initially check the connection flag that was set in the constructor and throws exception custom exception with required details.
- The method will then setup a connection to database and identifies all the contacts that met on the given date and adds them to a map as keys and value as false. The same set of contacts is also added to a list as well.
- It will then traverse through the list of above contacts and select a pair of contacts in the sequential manner from the list and adds them to the second list and a set after

checking that they have not yet been considered as part of any large gathering with the help of values in map.

- The method will then try to identify the contacts that met both the contacts of the above pair on he given date and then adds them to the second list and the set.
- If the number of contacts in the second set is greater than given minSize, then the method will proceed further to next steps. Else, it will iterate to next loop picking a different pair (go back to step 3).
- The method will then try to identify the number of contacts i.e., 'c' from the above set of contacts that have been in contact with each other for at least given minTime of duration.
- It will then try to calculate expression (c/((n*(n-1))/2)) where 'n' is the total number of contacts from above set.
- If the expression results in a value greater than given density then the large gathering counter is incremented by 1 and all the contacts of this large gathering are set to false in the map so as to avoid them processing in the next runs.
- The method will then try to move to step 2 and pick other pairs until all the pairs are traversed.
- Finally, the method will close all the streams and return the counter of large gatherings.
- This method throws the following exceptions along with necessary details:
    - InvalidInputException in case of any general issues
    - ClassNotFoundException in case of driver manager issues
    - SQLException in case of any issues related to SQL queries

In this version of the class, the queries are designed using a String. However, it can be replaced by prepared statements in the further versions of the program.

MainUI

- This class is usually maintained by Government and thus has the necessary access to update the path of configuration file in the source code that is used for creating the government object.
- It creates an array list of type MobileDevice to store the devices that are created as part of this session locally.
- It usually allows the end users to interact with this program by providing a menu of options which are follows:
    - Add a new mobile device.
        - Records the input as configuration file and create a new mobile device with the help of government object created in step 1.

- The new mobile device is then added to the array list of mobile devices.

- Add contacts to an existing mobile device in the current session or create a new mobile device and then add the contacts.
  - Displays the list of mobile devices (two devices per row) along with a serial number and prompts the user to select one among them to add contacts to.
  - However, in case of any empty list or if user wish to create a new mobile device and then add the contacts to it, they can enter -1 instead of the serial number of the devices displayed above.
  - If -1 is given as input, the method will take input as configuration file and then create a new mobile device and adds it to the array list of devices (this step is skipped in case the user selects the valid serial number).
  - The method will then get the contacts details from the user and adds it to the mobile selected device.
- Add positive test to an existing mobile device in the current session or create a new mobile device and then add the positive test.
  - Displays the list of mobile devices (two devices per row) along with a serial number and prompts the user to select one among them to add contacts to.
  - However, in case of any empty list or if user wish to create a new mobile device and then add the contacts to it, they can enter -1 instead of the serial number of the devices displayed above.
  - If -1 is given as input, the method will take input as configuration file and then create a new mobile device and adds it to the array list of devices (this step is skipped in case the user selects the valid serial number).
  - The method will then get the positive test details from the user and adds it to the mobile selected device.
- Synchronize an existing mobile device in the current session or create a new mobile device and then synchronize.
  - Displays the list of mobile devices (two devices per row) along with a serial number and prompts the user to select one among them to add contacts to.
  - However, in case of any empty list or if user wish to create a new mobile device and then add the contacts to it, they can enter -1 instead of the serial number of the devices displayed above.

        - If -1 is given as input, the method will take input as configuration file and then create a new mobile device and adds it to the array list of devices (this step is skipped in case the user selects the valid serial number).
        - The method will then synchronize the selected device with government's database.
    - Allow the agency to record a COVID-19 test to government's database.
        - The method will collect the details of the test and adds it to the database via government's object.
    - Find the number of large gatherings on a given date
        - The method will record the date, minSize, minTime and density and calls the respective method via government object to identify the number of large gatherings.
- The above menu is continued in a do while loop until the user wishes to exit by selecting the choice as 7.

InvalidInputException

- This class is used for implementing a custom exception by extending the 'RunTimeException' class.
- It consists of a constructor i.e., 'InvalidInputException(String message)' which is used for recording the message that is passed when a new exception is thrown by the other class.
- The method 'getString()' is used for retrieving the error message which was passed when the exception was thrown.

JunitTestCases

- This JUnit class defines the below set of test cases for testing the functionality of classes – MobileDevice and Government.
    - test_create_government_object
    - test_create_mobileDevice_object
    - test_add_contacts_and_sync
    - test_add_contacts_and_sync_with_positive_result
    - test_consecutive_sync
    - test_multiple_contacts_same_day
    - test_input_validations_mobile_device
    - test_input_validations_government
    - test_multiple_positive_tests

  - test_for_gatherings_one
  - test_for_gatherings_two
- The detailed description of each test case if described in the section 4.0.

1.7 Limitations

- The number of large gatherings may vary with the actual result unless the actual result considers the same pairs in the sequential order (just like this program did).
- The amount of time taken to read and process the data from database becomes more when the data inside the two tables keep increasing.
- There is no option to clear the contact once they have been entered. This may cause issue when a wrong contact is updated accidently though the call to record contacts is purely dynamic in nature.
- Also, the program may provide invalid results if the person who is contacted does not sync the data regularly with government database and thus may cause others (who got in contact with this person) to record incorrect results via mobileContact() method.
- In main method, the user is expected to know the device hash in order to add the contact to a mobile device.
- Since new connections are setup by mobileContact(), recordTestResult() and findGatherings() in order to avoid any crashes in connection during the program life cycle, this will ultimately increase the response time as new connections need to be setup every time when the method is entered.

2.0 Test Cases

2.1 Input Validations

MobileDevice

- Create a MobileDevice by passing a configuration file that does not exist (incorrect filename).
  throws IOException
- Create a MobileDevice object by passing a configuration file that is empty.
  throws InvalidInputException
- Create a MobileDevice object by passing a configuration file that does not contain data/parameters in the given format.
  throws InvalidInputException
- Create a MobileDevice by passing a configuration file as empty string and null value.
  throws InvalidInputException
- Create a MobileDevice by passing a configuration file that has special characters in file.
  mobile device is created successfully
- Create a MobileDevice by passing a configuration file that has duplicate device name and address.
  mobile device is created successfully
- Create a MobileDevice by passing a null as Government object.
  throws InvalidInputException
- Pass empty string as individual to recordContact().
  throws InvalidInputException
- Pass null value as individual to recordContact().
  throws InvalidInputException
- Pass a string of special characters as individual to recordContact().
  contact is recorded successfully
- Pass date<=0 to recordContact().
  Note: Date is started as 1 i.e., 01/01/2020 is considered to be day 1.
  throws InvalidInputException
- Pass duration<=0 to recordContact().
  throws InvalidInputException
- Pass empty string as testHash to positiveTest().
  throws InvalidInputException
- Pass null value as testHash to positiveTest().

throws InvalidInputException

- Pass a string of special characters as testHash to positiveTest().
  positive test recorded successfully

Government

- Create a Government object by passing a configuration file that does not exist (incorrect filename).
  throws InvalidInputException
- Create a Government object by passing a configuration file that is empty.
  throws InvalidInputException
- Create a Government object by passing a configuration file that does not contain data/parameters in the given format.
  throws InvalidInputException
- Create a Government object by passing a configuration file as empty string and null.
  throws InvalidInputException
- Create a Government object by passing a configuration file that has special characters.
  creates government object successfully
- Pass empty strings as initiator and contactInfo to mobileContact().
  throws InvalidInputException
- Pass null values as initiator and contactInfo to mobileContact().
  throws InvalidInputException
- Pass special characters as initiator and contactInfo to mobileContact().
  contacts synced successfully
- Pass empty string as testHash to recordTestResult().
  throws InvalidInputException
- Pass null value as testHash to recordTestResult().
  throws InvalidInputException
- Pass string with special characters as testHash to recordTestResult().
  valid data
- Pass date<=0 to recordTestResult() and findGatherings().
  throws InvalidInputException
- Pass a valid string, date and result to recordTestResult().
  test result is recorded
- Operate when minSize and minTime are passed as negative values to findGatherings().
  throws InvalidInputException

- Pass density as negative float value to findGatherings().
  throws InvalidInputException
- Pass valid data i.e., date>=1, minSize>=1, minTime>=1 and density to findGatherings().
  finds required gatherings

2.2 Boundary Cases

MobileDevice

MobileDevice(String configurationFile, Government Object)

- Operate when configuration file has required data only in a single line.
- Operate when configuration file has multiple lines to read the required data.
- Operate when configuration line starts with an empty line.

recordContact(String individual, int date, int duration)

- Call when you are reading it as the first contact.
- Call when the mobile device already contains a list of contacts and you're appending the new contact to the existing list.
- Call with individual as a string with a single character.
- Call with individual as a string with multiple characters.
- Call with individual as same mobile device's hash value.
- Call with date as 1.
- Call with date>1.
- Call with duration as 1.
- Call with duration>1.

positiveTest(String testHash)

- Call to report positive test for the first time.
- Call to report positive result for the second time or more.
- Call when testHash is a string with a single character.
- Call when testHash is a string with multiple characters.

synchronizeData()

- Operate when there is a single contact to synchronize with government server.
- Operate when there are multiple contacts to synchronize with government server.
- Operate when there is a single positive test to sync.
- Operate when there are multiple positive tests to sync.
- Operate when there is a contact with contact date as exactly before 14 days.
- Operate when there is a contact with contact date as same date of sync/current date.

Government

Government(String configurationFile)

- Operate when configuration file has required data in a single line.
- Operate when configuration line starts with an empty line.
- Operate when configuration file has multiple lines to read the required data.

mobileContact(String initiator, String contactInfo)

- Call with initiator as a string with single character.
- Call with initiator as a string with multiple characters.
- Operate when contactInfo contains a single contact.
- Operate when contactInfo contains multiple contacts.
- Operate when contactInfo contains a single positive test.
- Operate when contactInfo contains multiple positive tests.
- Operate when there is a contact who has been tested positive exactly before 14 days of the contact.
- Operate when there is a contact who has been tested positive exactly on the 14th day after the contact.

recordTestResult(String testHash, int date, Boolean result)

- Operate when the test is the first one to be added to the database.
- Operate when the test is not the first test to be added to the database.
- Operate when date is 1.
- Operate when date is greater than 1.

findGatherings(int date, int minSize, int minTime, float density)

- Operate when date is 1.
- Operate when date is greater than 1.
- Operate when minSize is 1.
- Operate when minSize is greater than 1.
- Operate when minTime is 1.
- Operate when minTime is greater than 1.
- Operate when there is a single large gathering to report.
- Operate when there are more than 1 large gatherings to report.
- Operate when there is a gathering with exactly minSize contacts.
- Operate when there is a contact in the gathering with exactly minTime duration.

2.3 Control Flow

MobileDevice

MobileDevice(String configurationFile, Government contactTracer)

- Create a MobileDevice by passing a valid configuration file that has only device name and address in the given format along with a valid Government object.
- Create a MobileDevice by passing a configuration file that contains only device name and address, but in a different format.
- Create a MobileDevice by passing a configuration file that has more parameters/data other than device name and address.

recordContact(String individual, int date, int duration)

- Call when the contact/individual already exists with same date, but with different duration.
- Call when the contact/individual already exists with same duration, but with a different date.
- Operate when the contact already exists with same date and duration.

synchronizeData()

- Operate when there are no contacts and positive tests to sync with government server i.e., only device hash.

- Operate when there are contacts and positive test results to sync along with device hash.

Government

Government(String configurationFile)

- The given details are valid and able to connect to the database.
- The given details are invalid and unable to connect to the database.
- Create a Government object by passing a valid configuration file that contains domain name of the database, username and password details (to access the database) along with schema details in a given format.
- Create a Government object by passing a configuration file that contains only domain name of the database, username and password details (to access the database) along with schema details, but in a different format.
- Create a Government object by passing a configuration file that has more parameters/data other than domain name of the database, username and password.

mobileContact(String initiator, String conactInfo)

- Operate when initiator is new and does not already exist in the database.
- Operate when initiator already exists in the database.
- Operate when contactInfo does not contain contacts to sync with database.
- Operate when contactInfo does not contain positive test result to store in the database.
- Operate when contactInfo does not follow a proper format of contacts (at least one contact is in a different format).
- Operate when contactInfo consists of duplicate contacts.
- Operate when contactInfo consists of contacts who are already present in the database, but needs an update such as date of contact, duration of contact and result of the contact.
- Operate when positive test result details in contactInfo does not match with any tests in the database.
- Operate when positive test result details in contactInfo matches with any one of the tests in the database.

recordTestResult(String testHash, int date, Boolean result)

- Operate when testHash already exists in the database with same date and result i.e., a duplicate test.
- Operate when testHash already exists in the database but with a different date or a different result.
- Operate when result is true.
- Operate when result is false.

findGatherings(int date, int minSize, int minTime, float density)

- Operate when minSize is 0.
- Operate when minTime is 0.
- Operate when density is 0.
- Operate when there are no contacts on the given date.
- Operate when there are contact(s) on the given date.
- Operate when a gathering is greater than density.
- Operate when a gatherings is less than or equal to density.
- Operate when there are no large gatherings to report.
- Operate when there are minSize contacts in a gathering but with less than minTime duration.

2.4 Data Flow

MobileDevice

- Call recordContact() before positiveTest().
- Call positiveTest() before recordContact() and Government.recordTestResult()
- Call synchronizeData() before calling recordContact() and after calling positiveTest().
- Call synchronizeData() before calling positiveTest() and after calling recordContact().
  Note: synchronizeData() before recordContact() and positiveTest() is mentioned as part of Control Flow.
  Note: synchronizeData() after recordContact() and positiveTest() is mentioned as part of Control Flow.

Government

- Call mobileContact() before recordTestResult() and findGatherings().
- Record a test in the server by calling recordTestResult() before mobileContact(), findGatherings() and MobileDevice.positiveTest().

- Call findGatherings() before calling mobileContact() and recordTestResult().
- Call findGatherings() after mobileContact() and before recordTestResult().
- Call findGatherings() after recordTestResult() and before mobileContent().

2.5 Additional tests

These test cases are mainly designed to test the required database schema.

- Insert data into contacts table and tests table.
- Update data in both contacts and tests tables.
- Insert duplicate rows into both contacts and tests tables.

3.0 Test Plan

- All the test cases specified in the section 2.0 have been tested using the helper JUnit class i.e., “JUnitTestCases.java”.
- Though few of the test cases were not separately created as a test, they have been tested as part of other tests present in the JUnit file.
- The following are the tests along with necessary details:
    - test_create_government_object
      test case designed for testing the creation of government object (server).
    - test_create_mobileDevice_object
      test case designed for testing the creation of mobile device object.
    - test_add_contacts_and_sync
      this test case is used for testing the scenario when contact(s) from a mobile device is added and synced with government server.
    - test_add_contacts_and_sync_with_positive_result
      this test case is used for testing the scenario when contact(s) from a mobile device is added and synced with government server where one or more of the contacts have been tested positive within the 14 days period from the contact. The outcome may vary if you are trying to sync for the first time or not.
    - test_consecutive_sync
      this test case is designed for testing the scenario when multiple syncs are performed consecutively. The outcome may vary if you are trying to sync for the first time or not.
    - test_multiple_contacts_same_day
      this test case is used for testing the scenario where same contact is made on different parts of the day either with same duration or a different duration.
    - test_input_validations_mobile_device
      this test case is used for testing the different input validations for class MobileDevice.
    - test_input_validations_government
      this test case is used for testing the different input validations for class Government.
    - test_multiple_positive_tests
      this test case is designed for testing a scenario where a contact has multiple positive tests recorded.

- test_for_gatherings_one
  this test case is designed for testing a scenario where more than one large gathering exists on the given date.
- test_for_gatherings_two
  this test case is designed for testing a scenario where no large gathering exists on the given date.

- The above tests provide a code coverage of 93.1% for MobileDevice class and 91.7% for Government class. Please note that the remaining instructions that are missed are usually associated to general instructions of the program. Also, these values will differ based on the data present in the database.
  Note: These coverage values correspond to tests that were executed when the tables were empty.

Thus, the above test cases (section 2.0) and test plan (section 3.0) helps to test the functionality of the program more efficiently in order to make it ready to deploy and run.