CSCI 3901

Software Development Concepts

Faculty of Computer Science

Final Project

Kishan Kahodariya B00864907

Assumption

- You will notice that each test case is taking significant amount of time to execute and this is because I have designed the code with the assumption that this solution will be used in real world, where each device needs to establish their own separate connection to the database while synching. For example, a device calls a method which wants to access the database, then it will first of all establish its own connection to the database and then perform the necessary operations. Once the changes are done, the connection is closed in each method. Doing so, ensures secure isolated access to the database per each MobileDevice and avoid unwanted open connection.
- It is assumed that variables date, duration, testDate used by different methods is always of positive integer type and not of date (DD/MM/YY) or time (HH:MM:SS) type. Also, if any of this variable is taken as 0, then input is considered invalid.
- When MobileDevice calls positiveTest() with a testHash, then it is assumed that if government object calls recordTestResult() for the same testhash, then the result should be positive i.e true as well, otherwise the record won't match in the database and generate error.
- Both config files – one for government and other for MobileDevices will be according to the format defined in the documentation. It is assumed that the config files will not contain blank lines, unnecessary white spaces and "" are not used to describe a string value.
- For now, the config files are being fetched from the config folder and path it shows is for my PC. You need to replace the paths with yours in each test case of testing framework to read your own config file. Note that config(1-10)*.properties are files for MobileDevices and Government config.properties is for government object..
- It is advisable to keep the contact data entry in the database mutual. For example, if A have recordcontact() with B, then B should also recordcontact() with A for the same day and time duration.
- The government object needs to be created successfully, otherwise error will be generated while creating MobileDevice object as well and no method will be executed as intended. If not followed, every test case in the testing framework will fail.
- testHash are case-sensitive and it is assumed that it will be of length 6.
- It is assumed that SQLscript will only be executed on the mentioned database – "**jdbc:mysql://db.cs.dal.ca:3306/kahodariya**" (included in the Government config.properties). Username and password are **kahodariya** and **B00864907**.

- It is assumed that for 2 same devices (A, B), A.recordcontact(B) is called for a specific day once, then if again A.recordContact(B) is called for same day but different time between those device, then it will be ignored. But if same devices calls recordContact() for different day, then it is considered valid and inserted as a new entry in the database.
- Note that if a device calls positiveResult() or recordContact(), then it is not updated or reflected in the database right away. It will only be updated when the device calls synchronizeData().

Database design

contactInformation Table

- This table stores every contact information from numereous MobileDevice and is designed to avoid duplicate contact entries.
- Table consists of:
    - **id** (primary key) – auto increment index
    - **Mobile_ID** – hash id of device which calls synchronizeData()
    - **Contact_ID** – hash id of device with whom contact is recorded
    - **NumOfDays** – date of contact
    - **Contact_Duration** – no. of minutes both device when in contact
    - **Test_ID** – test hash of Mobile_ID
    - **TestDate** – date on which the test was conducted
    - **Result** – result of test conducted

testDatabase Table

- This table stores data about each MobileDevice and unique testhash which is associated with it.
- Also, it acts as a buffer for tests records which are not yet associated with any MobileDevice (called using recordTestResult() by government object).
- In short, this table keeps tracks of test records for all MobileDevices, whether anyone has tested positive or not.
- Table consists of:
    - **Mobile_ID** (primary key) – hash code of MobileDevice

- **TestHash** – Unique test hash per Mobile_ID and of length 6
- **TestResult** – true or false
- **TestDate** – date on which the test was conducted

Code Design

MobileDevice class

Important Global Variable

- **device_Hash_ID** – unique hash value of Device
- **contactList** – a Map data structure, which acts as a local database for the device to store contact information with different devices before synching it with main database
- **contactTracer** – Government object to access main database using government class method
- **testPositiveID** – store unique hash value of length 6 for each device, if it has tested positive for COVID-19

MobileDevice(String configFile, Government contactTracer)

- In this constructor, **configfile** variable takes either file name or file path as an input and note that the code is only capable of processing file of **properties** type such as config.properties, device2config.properties, etc. For example, the file needs to following a format as below:

  "address=151.182.142.109\ndeviceName=Naruto"

  (**Note: Here "\n" denotes new line (refer files in config folder)**)

- If file of other format is passed, then "File not Found" message is displayed in the console. Talking about message, if error occurs while reading input from file then the message "Error while reading file" is displayed to notify user that they have some problem with the file.
- After successful reading network address and device name from the file, it appends them using "|" and then use it to calculate the hash value (length of 64)

of that particular device which will be unique in the database using SHA-256 encryption and is assigned to **device_Hash_ID** variable.

- **contactTracer** is Government object that is declared before creating any MobileDevice object (refer Testing Framework) and assigned to global variable **contactTracer**, which indicate that every MobileDevice has an object to interact with the main database.
- Note that this method will only read the file if the Government object is created successfully and **assignFlag** is set. Then, if MobileDevice object is also created successfully, then **deviceFlag** is set, which indicate that this object can now execute class methods.

boolean recordContact(String individual, int date, int duration)

- Initially, **assignFlag** & **deviceFlag** are checked, if they are set or not as they indicate if Government and MobileDevice objects are created or not. If not, it returns false.
- After that, each input is validated i.e. if **individual** string is of length 64 or not, **date** and **duration** are of integer type, non-negative & non-zero value because if it doesn't validate then "Invalid contact hash, date or duration of contact" message is displayed and method returns false.
- **Individual** parameter needs to taken as another MobileDevice's hash value i.e. using "**MobileDeviceObject.device_Hash_ID**" (refer testing framework for usage). Other than this, it be invalid.
- This contact information (individual, date, duration) along with MobileDevice's device_Hash_ID,is stored as an MobileDeviceData object and stored in **contactList** which acts as a local database for each MobileDevice, along with index value as key.
- Note that this contact will not be inserted to the main database (contactInformation) right away as it will only happen when the device calls synchronizeData().
- This method is designed such that if a device has called positiveTest() prior to any recording any contacts, then now if recordContact() in called, that particular test hash will be associated with each contact for future notifications.

boolean positiveTest(String testHash)

- Initially, **assignFlag** & **deviceFlag** are checked, if they are set or not as they indicate if Government and MobileDevice objects are created or not. If not, it returns false.
- This method allows to record test hash for positive COVID-19 test result for the MobileDevice, which is unique per MobileDevice and is assigned to the global variable **testPositiveID.**
- **testHash** is an alphanumeric string of length 6 and is only associated with one MobileDevice and is unique in the database. **testHash** is checked at the start if it is of the required length or not and whether the MobileDevice is already associated with another test Hash or not. If any of this conditions are not met, method returns message "Invalid TestHash" or “Device is associated with prexisting TeshHash” respectively.
- If the device contains list of contact in its local database (contactList), then the given teshHash is associated with them as well, for future references.

boolean synchronizeData()

- Initially, **assignFlag** & **deviceFlag** are checked, if they are set or not as they indicate if Government and MobileDevice objects are created or not. If not, it returns false.
- After that, connection to the database is established using Government object **contactTracer** to call **establishConnectionToDatabase()** which establish connection to the database before synching contact information from MobileDevice with main database. If there’s any error connecting, “Problem Connecting to Database" message is displayed.
- Once connection is established, **mobileContact**() from government class is called using **contactTracer** to insert or update respective contact information for different contacts stored in contactList to both tables – contactInformation & testDatabase.
- This method is also capable of synching a specific type of data, where MobileDevice only have information about its positive Test Hash and nothing regarding having contact with any other MobileDevices. For example, if

MobileDevice have only called positiveTest() before executing synchronizeData(), then also the database will be updated only with this information as well.

- Once every contact from contactList is synched with the database, the connection to the database is closed at the end, and the method returns **true** if the MobileContact() returns **true**, which indicates that the MobileDevice have come in contact with a COVID-19 positive device or returns **false** if no such contact is made.
- Note that the method only returns true if the contact has positive test result and also the difference between date of contact and date of test is less than or equal to 14 days. If either of the condition is false, then the method returns false.

toHexString(byte [] hash)

- This method is used internally by the constructor only, to store the new hashed value in device_Hash_ID variable.

Government Class

Important Global Variable

- **deviceList** – stores hash of every contacting MobileDevice
- **allLoginInfo** – used as a flag and is set when config file is successfully read

Government(String configFile)

- In this constructor, **configfile** variable takes either file name or file path as an input and note that the code is only capable of processing file of **properties** type i.e. Goverment config.properties which is located in config folder. For example, the file needs to following a format as below:

  "database=jdbc:mysql://db.cs.dal.ca:3306/kahodariya
  user=kahodariya
  password=*********"

- Note that to access the main database, use only the provided file **Government config.properties** from the config folder, as it contains all required login information.

- After this, the config file is read and data from the file is stored in **database**, **username** and **password** variable and if all these is read and if any error occurs then "Error occurred while reading Gov. config file".

boolean mobileContact(String initiator, String contactInfo)

- In this method, **initiator** denotes the hash code of the contacting MobileDevice (from MobileDevice.synchronizeData()) and **contactInfo** denotes all the contact information of the contact object from MobileDevice's local database.
- From this, data - Mobile_ID, Contact_ID, NumOfDays, DurationOfContact, Test_ID and Result, whichever is available in the string, is assigned and inserted as a new entry and code is design such that if database already contains data of same Mobile_ID, then whichever information is not available in the database, only that data is updated in both tables which helps to avoid inserting duplicate data in the database and reduce the time for traversing the whole database.
- If any error occurs, "Not able to synch data with the Database" message is displayed.
- Once inserting new data or updating current data is done, synchDatabase**()** which is an internal method, is called which synchronizes newly inserted data between both tables and also checks if there was any contact with COVID-19 positive device, and if yes, then it return **true** otherwise **false**.
- mobileContact() returns **true** or **false** based on what **synchDatabase()** returns.

recordTestResult(String testHash, int date, boolean result)

- At first, length of **teshHash** is validated i.e. if it is not of size 6, and also, date is checked, if its non-zero & non-negative value. If either of this condition is not met, the method display message "Check your TestHash or Date" and returns false.
- Once they are validated, it checks if the flag allLoginInfo is set or not, and if it is then **establishConnectionToDatabase()** is called to secure connection with the database using login details read by the main constructor.
- Once connection is made, the database (**testDatabase table**) is checked if it contains data with **testhash** or not. If it does, then only the **date** and **result** (if not present in the table) is updated to avoid entering duplicate data. Note that each testhash is unique per MobileDevice, so while inserting if the testHash is not

associated with any Mobile_ID, then it is inserted as a new entry with “default_device” as its Mobile_ID and in the future, if there is an entry of any Mobile_ID which is associated with this testHash, then that device is linked to that preexisting teshHash automatically.

- If any error occurred while updating the database, then "Error in recording the test result" message is displayed.
- Once necessary updates are made to the tables, the connection to database is closed to ensure security and avoid unwanted access to the database.

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

- Initially, date and minTime variable are validated, whether these are **non-negative** and **non-zero** or not. Also, density variable should be **a non-negative** float value and minSize should be also **non-negative** value. If either one doesn’t satisfy the condition, then error message "Invalid input. Check your date, minSize, mintime or density “is displayed and the method returns 0.
- After validation it checks if the flag **allLoginInfo** is set or not, and if it is then **establishConnectionToDatabase()** is called to secure connection with the database using login details read by the main constructor.
- Then, those contacts pairs are fetched from the database using SQL query who have made contact on **date** and for at least **minTime**.
- Once this is done, the set of individuals (**S**), number of appropriate pairs (**c**) and **m** value is figured out as per instruction and then the final **c/m** ratio is calculated for each gathering. If this ratio is greater than **density**, then **numberOfLargeGathering** is incremented.
- This is repeated for each possible gathering, and at the end, the method returns numberOfLargeGathering as an output.

establishConnectionToDatabase()

- This is an internal method which is used to establish connection with the main database using login details, extracted from government’s config file.
- If an error occurs due to problem in jdbc driver, then “Error in jdbc Driver” is displayed and connection to database fails, then "Error establishing connect to the database" is displayed.

boolean synchDatabase(String deviceID)

- As mention earlier, this method is designed to synch data between both tables – contactInformation & testDatabase.
- This method is called at the end of MobileContact(), which means that during execution, if either of the table is updated with new values from MobileDevice, then releavant changes are made into contactInformation based on updates made into testDatabase and vice versa.
- At the end, method checks if **deviceID** (which denotes hash Id for particular MobileDevice) has been in contact with another COVID-19 positive device or not. If it has, then returns true or else false.
- If there's any problem while updating the database, then "Problem while synching database" message is displayed.

List<Pair> findPairs(List<Pair> pairs) and

boolean findPairWith(List<Pair> pairs, Pair pair)

- This both methods are only utilized internally by findGatherings() method.
- FindPair() class is used to perform internal comparison within list of pairs.

boolean clearDataBase()

- This method is used to clear the database i.e. both tables, before inserting any values and executing any test cases.
- This method is only used in the testing framework (Testing.java).

MobileDeviceData Class

- This class is used as data object which stores contact information for a MobileDevice, when it has contact with another MobileDevice.
- Information includes Mobile_ID (hash id of current device), Contact_ID (hash id of other device), NumOfDays (date of contact), DurationOfContact, Test_ID (positive test hash if is available), DaysOfTest (date of Test) and Result (result of that test).

- Class contains a basic constructor to create an object (used in recordContact() of MobileDevice), setter methods to set variables separately if need arises and contactInfo() which is utilizes in synchronizeData() of MobileDevice.
- Basically, contactInfo() aggregates contact information of the contact object () from local database (contactList) of the MobielDevice and generates a single string which is then passed to MobileContact() as input, which is called using government object (contactTracer) in synchronizeData().

Pair Class

- Pair Class is used as a pair object which stores Mobile_ID and Contact_ID as single object, which are obtained using the SQL query executed at the start of findGatherings() method for a specific date and time.
- Based on operations using these pairs, the gatherings are calculated and reportedly accordingly.

Test Plan

- **Testing.java** is an JUnit based file testing framework and is located in **src** folder.
- In this file, the **ValidationForMobileDeviceMethods** class perform validation for different methods of MobileDevice class. Along with validation, boundary cases for most input variable is also tested by considering min and max value for that particular variable.
- The type of validation done are listed below:
    - 1^{st} ,2^{nd} – test correct and incorrect file path for MobileDevice object
    - 3^{nd} - Whether the device Hash code is of appropriate size
    - 4^{th} - test recordContact() with all correct value
    - 5^{th} - teshHash for positiveTest() is either empty string or is of length less than 6
    - 6^{th} - teshHash for positiveTest() is valid
    - 7^{th} (Min. Boundary Class) – use min. possible value of string, integer for recordContact()
    - 8^{th} (Max. Boundary Class) - use max. possible value of integer and a valid string for recordContact()

- Next, the **ValidationForGovermentMethods** class perform input validations for methods of Government class. The type of validation done are list below:
    - 1st – test correct or incorrect file path for Government object
    - 2nd – test recordTestResult() with all valid input
    - 3rd – testHash is either empty or of wrong length for recordTestResult()
    - 4th – date variable is either negative or 0 in recordTestResult()
    - 5th (Min. Boundary Case) – testHash is empty and date is a negative value
    - 6th (Max. Boundary Case) – testHash is valid string and date is a max. possible value
    - 7th - Correct inputs for findGatherings()
    - 8th (Boundary Case) – max. an min. value for each variable of findGatherings()
- At last, **TestCases** class includes total 9 cases which varies in term of order in which different methods are called and how output of each method varies with change in order of their execution.
- In first test case, multiple device calls recordContact() in group and after that respective devices calls synchronizeData() in the end. Each synchronizeData() will return false, no device has reported positive test result.
- In 2nd test case, M1 and M2 device records contacts with each other simultaneously and after that M1 calls synchronizeData() and M2 reports positive test. After this, M1 again calls synchronizeData() but instead of returning true, it returns false. This is because the positive testHash of M2 doesn't have any testDate linked to it, so it's hard to determine the 14 days' criteria and whether M1 was in contact with M2 while it was COVID-19 positive.
- In 3rd test case, unlike second case, the positive testHash of M6 does have a test date and the difference of days between contact date of M6 with M5 and testDate is less than 14. So, when M5 calls synchronizeData(), it is been notified that it has come in contact with a positive device. Also, based on contacts recorded, large gatherings are checked.
- 4th test case tests different methods being called at random multiple times and its accuracy to insert or update database or return any error and return true if any positive contact is made.
- 5th test case checks if device has called positive test before recorContact() and then synchronizeData(), whether it is able to those positive testHash with those contacts and insert them in to database.

- 6th test case, checks if methods called in order – recordTestResult(),positiveResult(),recordContact(), synchronizeData(), is able to link relative data together and insert into database or not.
- 7th and 8th test case checks execution of methods in different order, synchronizing data after just one new information, validation of input data and is the system able to function properly when multiple devices call synchronizeData() simultaneously.
- 9th test case, checks if findgatherings() is able to report any large gathering based on contact details recorded and synched with the database.