Name: Ridampreet Singh Jaggi

BannerID:B00882657

Project Title: Covid-19 contact tracing application

Goal: The course project is your opportunity to demonstrate all of the concepts from the course in one body of work.

Assumptions:

1. Before starting the application the Government object as well as all the objects of the Devices that will interact need to be initialized with the file path which contains their names and address using the respective Constructors. This is because this application encrypts the device info as soon as it gets the name and the address of the file and then each device can be recorded in contact with the other using the deviceHash that is generated after the constructor for MobileDevice Class has been called.

2. When device A is recording that it contacted device B, the same thing should happen from device B. For example:

   A.recordContact(B.deviceHash)
   B.recordContact(A.deviceHash)

   This is done to ensure the consistency of the Contacted table which contains the details of a meeting.

3. When a device records that it has tested positive for Covid using the positiveTest method, the information is packed into a string and is not passed to the database until the device calls synchronizeData().

4. When running the JUnitTest class I have made sure that database tables do not contain any data, meaning that tables should be available but no data in the tables should be present, for this a function cleanUpTheTables has been made.This function is executed first to clear all the entries to ensure correct working of JUnit.

5. Whenever there are some error conditions reported from the methods, i have printed them on to the console.This approach was taken because in the real world applications we need to inform the user about the error so that the human can correct it and then access the app again for example errors like connection and file path wrong etc.

6. There can be no ‘;’ and ‘/’ in the names of contacts, testHashes or any kind of information entered by the user, this is because I have performed splitting based on these signs. Also there can be no special characters that are included in the input as this may create errors.

7. testHashes must be entered without any special characters.

8. For findGatherings, we do remove duplicate gatherings because there is no benefit from counting the same gathering more than once, but when a sub gathering occurs it is counted as a different gathering for example: A,B,C is different than A,B,C,D. This is done so that we alert the government in as detailed way as possible. Moreover by this method it will be easy for them to understand how a sub gathering led to a huge gathering.

Explanation of the solution:

Every method outlined in the project pdf provided for this project has been explained, for reference to particular methods, you can see a detailed explanation below.

1. The format of the config file that contains information about the database is:

database= jdbc:mysql://db.cs.dal.ca:3306/rjaggi

user= rjaggi

password= B00882657

and the format for the files from which the device information is fetched is:

address=127.0.0.2

name=Iphone8

These are then retrieved using the properties file reading process in the constructors which fetch information from keys like **database**, **user**, **password** etc. These files are available in the resources folder and can be used for testing.

2.Starting with the **Government()** constructor, the constructor takes a parameter as a path to the config file that needs to be read in order to create a connection with the database. Upon successful reading of the file a connection is established

3. **MobileDevice()** constructor is called in order to register a device for this application, in the real world we can see this process as installing the app from play store or app store. The constructor first reads the path provided to it and then stores the Government constructor in its own class which is later used to transfer

information. The constructor when reading the information about the device sends the information to the encryption function which further encrypts the device info using SHA-256 algorithm. This device hash is stored in a public variable in the MobileDevice class, the reason this variable is public is because it will be used later by other devices to record contact.

4.**encryptDeviceUser():**This method is used to encrypt device information, it is called from the MobileDevice method. The method uses SHA-256 encryption

5.Now that we have successfully established a connection and have registered all the devices, we need to start recording contacts.

6. The **recordContact()** method, a part of the MobileDevice class, gets the details of the contacts the current device has interacted with and then stores it in a map which will be later used to send the info to the government class. There is a map of class covid_test_results being used to store the information of the contact. This map contains key as a String which refers to the deviceHash of the individual the device has come in contact with and the value is an array list which contains the deviceHash of the device who came in contact, separated by ‘,’ is the date and the duration .This list of objects is then later iterated in the synchronize method to create a string which will be passed to the government to store these meeting details in the database table Contacted.

7. The **positiveTest()** method adds the testHash reported by the device to a String while each testHash has been separated by the symbol ‘/’. This will later be used to separate the testHashes.

8.**recordTestResult():**This method is used to record the testHash, the result and the date of the test. The method records all the testHashes received by the government and then enters into the database which is done by method sendResultToGovt() method which simply makes a query to insert the testHash.

9.**sendResultToGovt():** This method inserts the testHashes to the database if the testHash provided has not been already recorded in the database. If it already exists we do not do anything.

10.**sycnhronizeData()** method packs the information and then sends it to the method in the government class, which then enters the information into the relevant tables. There are 2 additional methods that I have created to handle special case which will be invoked from the synchronizeData in cases such as a device has not come in contact with any device and has not reported any positive testHash, this will be achieved by the method named insertForNoContactAndTestHash(deviceHash) which accepts the parameter as a deviceHash only because since no contacts and no positive testHash has been reported by this device, we just insert this device if its not already present in the Users table. The second case for which I have created a method that is invoked by the synchronizeData is the case where there have been no devices that have been near this device but the device wants to report a positive testHash. In this case the method insertForNoContactsButTestPositive(deviceHash,arrForSendingToTheFunc) which accepts the deviceHash for the device which wants to report the testHash and the arrForSendingToTheFunc which has the testHash and then it is entered into the database. Rest of the cases that report the contacts but no testhash and contacts and testhash both are handled by entering data into the mobileContact().

11. Information from the **synchronizeData()** method is passed in the form of a string which contains the devices recorded separated by their date and duration using the ‘,’ and this separated by ‘;’ to indicate other devices that came in contact. This string is then separated by ‘/’ which marks a testHash reported by this device, the detailed version of this format can be found below:

DeviceA has interacted with three devices i.e. B,C,D each separated by ‘;’ and it also has reported that it has tested positive for three Covid tests which are separated by the ‘/’ sign.

deviceB,date,duration;deviceC,date,duration,deviceD,date,duration/CovidTest1/CovidTest2/CovidTest3.

The string is then further split based on ‘/’ and ’;’ to get the contacted devices in the beginning of **mobileContact** method, when the string is split by ‘/’, an array is formed with all the contacts on index[0] and all of the testHashes at the following indexes, the index[0] is further split by ‘:’ to get each contacted device, the array is further sent to map the user of the test with the testHash in the covid_test_result table.

There are methods that are called within synchronizeData for different conditions as follows:

1.When there are no contacts and no testHashes reported by the user.(insertForNoContactAndTestHash is called)

2.When there are no contacts but there are some testHashes that need to be reported.(insertForNoContactsButTestPositive is called)

3. When there are contacts but no testHashes that need to be reported(mobile Contact() is called)

4. When there are contacts and testHashes that need to be recorded(mobileContact() is called)

12.**mobileContact():** This method takes up the string passed by the synchronizeData as a string and then performs the task of breaking the array by splitting the recorded contacts using ';' .While we record the contacts, we check if the same meeting has happened before, meaning that if the same contacts have met on the same day, if yes we just update the meeting details with the old duration plus the new duration by calling the method **checkIfSameMeeting()**.Before ending the program we send the array we have received to the method **resultIdentification** which will start mapping users to the testHashes in the database(more on this method later). There are two scenarios where mobileContact() is called.

a. When the device that has called synchronizeData method and has some devices that came in its contact and some positive testHashes.
b. When the device that has called synchronizeData method has some devices that came in contact but it does not have any testHashes to report.

In case where there are testHashes that need to be reported from the device are present, that is case **a,** we set a flag for this and this is used to report the testHashes separately by calling resultIdentification() at the end of this method.

In cases where there are only contacts that need to be reported, we just insert the device in the users table. This is case **b**.

13.**insertForNoContactAndTestHash():** This method is used to insert just the device as this will be called when there are no meetings or no testHash that need to be reported.

14.**insertForNoContactsButTestPositive()**:This method is called within the synchronizeData. The method only inserts the device if not in the table and the testHashes are reported by calling the resultIdentification method.

15.**checkIfSameMeeting():** This method is used to update the meeting duration as it was called because it was found that there exists a meeting on the same date between the same devices and at the same day. This simply updates the duration of meeting to the old plus new duration.

16.**resultIdentification():** This method is called at the end of the mobileContact(). The method takes input as the initiator device and the s1, the s1 contains contacted devices and the testHashes separated by the '/'. The method then starts iterating the s1 array from index 1 to length-1, the reason we start from index 1 is because index 0 contains all the contacts and after that all the indexes contains testHashes till the end. We then check if testHash is already present in the database, if yes then we just map the UserID of this deviceHash to the test.

17. **findGatherings()** method is used to find gatherings on a particular day which occurred between a minSize of individuals for minTime and then calculates the density based on this data to decide whether this can be counted as a gathering or

not. The first step this method does is to fetch all the users that met each other at that day in the form of pairs, this set is called allOfTheMembersAtThatDay in the code, and then we fetch the pairs of user that met that day for minTime(this will be later used to make decisions on whether the gathering should be a part of the gathering or not), this set is by name tempForNew in the code. We then start iterating the list of pairs that are present in the set allOfTheMembersAtThatDay, we get each pair that met at that day and then we find the intersection of individuals this pair met at that day we then calculate the size of this set and if the set size is more than the minSize, we perform manual pair forming by iterating and making the combinations of pair from the set for example a set for pair **AB** contains elements-[**ABCD**](included A and B to make pairs), then the set for this set would look like this->**AB,AC,AD,BC,BD,CD**. Upon getting these pairs we pick each of them and then check in the tempForNew set which contains the pairs that met at that day as well as for minTime, please not that every pair in every set has been stored in a sorted manner and the pairs i am referring here as A and B would actually be UserIDs of the devices like pair (198,199), here A and B represent 198,199. After counting the pairs that met that day for minTime, we move on to the next part that is calculating the max pairs from individuals by the formula provided in the pdf provided by the professor, this is stored in the variable **m,** we also have the count of the pairs that met that day which we took out from iterating and identifying common pairs from manually formed pairs and the pairs present in the tempForNew we name this count as **c.** The calculation of **c/m i**s performed and then we check if this is greater than the density provided to us in the findGatherings method, if yes we count it as a gathering and if no, we skip this gathering. The total count of gatherings is stored in the gatheringCount and is returned once all the pairs have been checked. In cases where an error occurs in the method or there are no gatherings that day we return 0. Please note that while returning gatherings, i have not considered to report a gathering which has already been reported for example if gathering **A,B,C has been reported the gatherings such as B,A,C or C,A,B** will not be reported back. In cases where sub gatherings have been reported such as **A,B,C , I do consider reporting a gathering which contains another element in addition to these such as A,B,C,D, so A,B,C and A,B,C,D would be considered two separate gatherings.**

18. **check_for_alert():** This method is used to check if the user has interacted with any of the covid positive people. We consider that if the meeting happened 14 days before the other person tested positive or after 14 days the other person tested positive, we then check if the testHash because of which the initiator of synch will be returned true is new, meaning that the created time of the test is greater than the last synch of the device, this has been done so that only new covid-19 records are reported back. Once we know that this is the case we update the last sync time of this device to now(), which will again be used to compare that if the testHash this device is checking is recently being reported or is already among the old ones that have already been reported. After the last sync has been updated the person

currently synchronizing gets the return value from synchronize as true. This method is called by every method that is called in the synchronizeData.

Database Design:

Tables:

1.Users

2.Contacted

3.covid_test_results

Please note that while creating tables, we need to create the Users table as it provides UserID as a foreign key to the other tables.

Description of each table:

1.Users: This table has fields

a. UserID-Unique for every device and refers to one device only,this id is used everywhere in the db in other tables to refer to this device as reading ids is easier then reading long deviceHash.
b. deviceHash-Encrypted information of the device
c. lastSynch-Contains the last synch of the device with the database to check if it has been near any covid +ve patient

2.Contacted:This table stores information about the meetings:

a. ContactedID-auto, is unique for every meeting
b. sourceFkID-represents the source of meeting for example if A is recording that it met device B, then source would A and vice versa. This is a foreign key with the UserID of the Users table
c. contactedFkID- the user id of the device that contacted with source,This is a foreign key with the UserID of the Users table
d. contactDuration- stores the duration of the whole meeting

e. contactedDate- stores the date of the meeting, the meetings are stored in actual date format, we add the given days to january 1st 2021, this is done so that when we see the date of the meeting we can see the real date like we do in the real world

3.covid_test_results: stores the details of the testHash(covid positive results), has the following columns

a. UserID: Represents the device that has reported itself as true.
b. Date: the date of this test
c. Result: 1 means +ve and 0 means -ve
d. testHash to store the testHash
e. createdTime-real time of the moment this record was inserted, used to achieve the feature of fetching only new covid positive records in synch.

Queries used to create tables in database.

```
use dtabasename;

/**

Query below is for creation of the User table.This should be run first because other tables

are dependent on this due to the foreign key

**/

CREATE TABLE `Users` (
  `UserID` int(11) NOT NULL AUTO_INCREMENT,
  `deviceHash` varchar(255) NOT NULL,
  `lastSync` datetime DEFAULT NULL,
  PRIMARY KEY (`UserID`),
  KEY `personHashIN` (`deviceHash`)
);


/**

Query below is for the creation of the covid_test_result table which stores the test results of people.
```

```
**/


CREATE TABLE `covid_test_results` (

  `ID` int(11) NOT NULL AUTO_INCREMENT,

  `UserID` int(11) DEFAULT NULL,

  `date` date DEFAULT NULL,

  `result` tinyint(4) DEFAULT NULL,

  `testHash` varchar(45) DEFAULT NULL,

  `createdTime` datetime DEFAULT NULL,

  PRIMARY KEY (`ID`),

  KEY `UserID` (`UserID`),

  CONSTRAINT `covid_test_results_ibfk_1` FOREIGN KEY (`UserID`)
REFERENCES `Users` (`UserID`)

) ;


/**

Query below is for the creation of the contacted table which stores the meetings and
the details of the meetings between people.

**/


CREATE TABLE `Contacted` (

  `ContactedID` bigint(20) NOT NULL AUTO_INCREMENT,

  `sourceFkID` int(11) DEFAULT NULL,

  `contactedFkID` int(11) DEFAULT NULL,

  `contactDuration` int(11) DEFAULT NULL,

  `contactedDate` date DEFAULT NULL,
  `deleted` tinyint(1) unsigned zerofill DEFAULT 0,

  `createTime` datetime DEFAULT NULL,

  `createUser` varchar(255) DEFAULT NULL,

  `updateUser` varchar(255) DEFAULT NULL,

  `updateTime` datetime DEFAULT NULL,

  PRIMARY KEY (`ContactedID`),

  KEY `contactedFk` (`contactedFkID`),

  KEY `sourceIDIn`
(`sourceFkID`,`contactedFkID`,`contactedDate`,`contactDuration`),

  CONSTRAINT `contactedFk` FOREIGN KEY (`contactedFkID`) REFERENCES
`Users` (`UserID`) ON DELETE NO ACTION ON UPDATE NO ACTION,

  CONSTRAINT `sourceFk` FOREIGN KEY (`sourceFkID`) REFERENCES `Users`
(`UserID`) ON DELETE NO ACTION ON UPDATE NO ACTION

);
```

Test Plan:

To test that the program is robust the following tests were conducted and they have been categorised under categories. First while putting the wrong or incorrect file path name I put in the Government constructor, then observe the working and then document it, similarly for MobileDevice. For recordContact I have tried putting the other devices hash as null and then made sure that this meeting is not recorded. Same happens if the date is negative and the same happens if the duration is less than 1 .

The JUnit submitted along the project has a code line coverage of **81%** for the class Government and **93%** for the class MobileDevice and Class and Method coverage of **100%** for both the classes. This means that the majority of the lines of codes are being covered in the JUnit test, the reason why **81%** of lines are being covered in the Government class is because there are some exceptions which I am not invoking in the JUnit class

Input Validation:

Government():

1. Wrong file path

   Expected result:The error message is displayed on console

2. File path as null

   Expected result: The error messages will be printed to the console

3. File Path is empty

   Expected Result: The error message is displayed on console

MobileDevice():

1. Wrong file path

   Expected result:The error message is displayed on the console. If the device

   Has some kind of implementation then that related error will be printed.

2. Null as file path:

   Expected result: The error message is displayed on console

recordContact():

1. Null as the contacted deviceHash

   Expected Result: Meeting will not be recorded.

2. If the duration is negative or 0

   Expected Result: The meeting will not be recorded.

3. If the date is negative

Expected Result:The contact will not be recorded.

positiveTest():

1. Test hash as null

   Expected result: the testHash will not be recorded.

2. If the testHash has already been recorded and mapped to this device user.

   Expected result: No change happens.

findGatherings():

1. Date for which no meeting has been recorded

   Expected result: Total gatherings 0.

2. minSize as negative number:

   Expected result: The gatherings having individuals more than the minSize will reported.

3. Density as a negative number

   Expected result: All possible number of gatherings.

synchronizeData():

1. If the contact that is being synched has some incorrect details in them like discussed in the recordContact method above.

   Expected result: false

Boundary Cases:

recordContact():

1. Date as 0

   Expected result: Works normally

2. Duration as 1

   Expected result: Works normally.

recordTestResult():

1. Date as 0

   Expected result: Works normally

synchronizeData():

1. If meeting happens within 14 days of the test meaning that +14 days or -14 days

   For example: A met B at day 130 for duration 3 and B met A at same day for same duration

   and

   A met C at day 158 for duration 3 and C met A at same day for same duration

   A reported to be positive on day 144 and then calls synchronizeData(), and B and C also call synchronizeData()

   Expected result: on sync of B and C they both return true.

2. If meeting happens beyond 14 days of the test meaning that more than+14 days or similarly for -14 days

   For example: A met B at day 129 for duration 3 and B met A at same day for same duration

   and

A met C at day 159 for duration 3 and C met A at same day for same duration

A reported to be positive on day 144 and then calls synchronizeData(), and B and C also call synchronizeData()

Expected result: on sync of B and C they both return false.

findGathering():

1. Date as 0

   Expected result: will get gatherings for that day if present

2. Duration as 0

   Expected result: will get gatherings that satisfy the minTime duration and other constraints.

3. minSize as 0

   Expected result: will get gatherings for that day which have more individuals

   than 0.

4. Density as 0

   Expected result: will get gatherings for that day which have density more

   than 0

Control flow(Order wise starting from 1):

1. **Government():** To setup the connection with the database by providing url,name and password to it.
2. **MobileDevice()**: To initialize all the devices that will be recorded as contacts with each other
3. **recordContact():** To record that device A met device B.
4. **recordTestResults():** Method called to make entries into the database about the testHash and its result.
5. **postiveTest():** To record that the device has tested positive for Covid-19.
6. **synchronizeData()**: To pack information in a package and call the method that syncs data with the government database. If the device has been around anyone which was positive and also satisfies the 14 day period, then we return true, else false, this is done by calling functions from inside of mobileContact.

7. **mobileContact():** Method synchronizeData calls mobileContact() within itself to send data to the database when the device has reported some contact meetings and in some cases positive test hashes.Check for if the device has met any one positive by calling check_for_alert.
8. **insertForNoContactAndTestHash():** Method created by me to handle the synchronization of data in some special cases where the device neither reports any contact meeting nor any positive test hash, this method then directly enters the user to the database..Check for if the device has met any one positive by calling check_for_alert.
9. **insertForNoContactsButTestPositive():** This method is created by me in order to handle another special case where the device has reported no contacts that it has met but has reported the positive test hash..Check for if the device has met any one positive by calling check_for_alert.
10. **findGatherings():** Method used to find gatherings for a given date, minimum size, minimum time of meeting or duration of meeting and greater than the density entered.

Data Flow:

1.Calling MobileDevice() before Government()- This will produce error as the government constructor needs to be passed in the class MobileDevice()

2. Calling the recordContact method to record a meeting with a device that has not been initialised using the MobileDevice() will produce an error message.

3. Calling the findGatherings method before recording any contact will give gatherings as 0

4. Calling the method synchronizeData before recording any contact or the testHash, we get return value as false.

5. Calling the recordTestPositive of Government before calling the positiveTest method of the MobileDevice() will insert the testHash normally in the table.