CSCI 3901 Winter 2021
Final Project Report

The CSCI 3901 final project is a working implementation of the COVID – 19 contact tracing application. The project is made using Java & DBMS components. This report explains the Assumptions, Implementation Details, Database Design & Test Plan.

Assumptions:

1. **Exception handling assumptions:**
    - **a.** I am throwing various exceptions for input validations & other errors faced in the code like IllegalArgumentException, FileNotFoundException, SQLException, IOException etc.
    - **b.** I am also passing a customized message with each thrown exception.
    - **c.** But I am catching all these exceptions in the RuntimeException and handling them & printing the customized message passed with the exception using **getMessage ()**.

2. **Hashing Algorithm:**
    - **a.** The hashing algorithm used is of SHA-256 using Message Digest class.

3. **Database related assumptions:**
    - **a.** Before running the program, the **FinalPorject_Create.sql** file needs to be run that has the code to create the database schema.
    - **b.** There is another SQL file **FinalProject_Drop.sql** that will drop the tables in the schema & another file named **FinalProject_Embeded.sql** that has all the queries that I have used in the java program.
    - **c.** I have made a public method **resetDatabase ()** in the Government class, that basically connects to the database & drops all the tables if they exist & then create the tables.
    - **d.** **So before running any test case I have made a method call to resetDatabase () method to reset the data to get accurate results.**
    - **e.** The **recordTestResult ()** method of the Government class needs to be called before the **positiveTest ()** method of the MobileDevice class. So, to maintain consistency in the database, the design of the database is done in such a way that if **positiveTest ()** method is called before **recordTestResult ()** & then **synchronize ()** method is called then data won't be inserted into database due to foreign key violation constraint.
    - **f.** Each & every time I need to run a query, I open the connection to the database & then I close the connection to the database as soon as the query is run, within the same method. Thus improving the efficiency.

4. **Config file relates assumptions:**
    - **a.** The config files passed to the MobileDevice & Government class can be of **.properties/.txt/ without any extension.** The config file with any other extension will cause the program to throw a RuntimeException.
        - **i. MobileDevice class:** The data in the config file for MobileDevice class should be as follows, there should not be any blank lines or white spaces & the contents are case-sensitive;
          address=127.10.255.0

deviceName=Cosmos

Exceptions thrown: The constructor will throw a RuntimeException displaying the error message. Exceptions are thrown in following cases:

1. The format of the contents in the text file is not valid
2. The file does not exist
3. The contents of file are empty
4. The file name contains some other extension

All configuration files that I used for testing have been pushed to the GitLab repo.

ii. **Government class:** The data in the config file for Government class should be as follows, there should not be any blank lines or white spaces & the contents are case-sensitive;
database=jdbc:mysql://db.cs.dal.ca:3306
user=drshah
password=B00870600

Note: While connecting to the database I am appending the user behind the database string, instead if directly taking the database with the user at the end

Exceptions thrown: The constructor will throw a RuntimeException displaying the error message. Exceptions are thrown in following cases:

1. The format of the contents in the text file is not valid
2. The file does not exist
3. The contents of file are empty
4. The file name contains some other extension

All configuration files that I used for testing have been pushed to the GitLab repo.

iii. **The program is designed in a way that if any MobileDevice class instance is deleted in between while the program is still executing then, all the data for that MobileDevice would be lost i.e. all the contacts & test hashes.**

iv. For Government class I have implemented Singleton Design Pattern, so at any point in time only one instance of the Government class be created and used.
    1. To implement the Singleton Design Pattern, I had to make the Government class private and instead created a **public getInstance ()** method.

v. The date given as input is the number of days since 1st Jan 2021. I am taking this input & converting the **noOfDays** into the actual date of format **yyyy-mm-dd,** it is calculated inclusive of the start & end date. If **date = 0,** then I return 2021-01-01 (as I am starting from 1st Jan 2021).

vi. The date difference that I have taken between the **Contact_Date** & **Test_Date** in the embedded SQL query is the Absolute difference.

Database Design:

The database design for the COVID – 19 contact tracing application consists of 4 tables:

- **MobileDevice**

- **TestResult**
- **MobileDevie_TestResult**
- **Contact**

Note: To perform insertions & select queries to the database & even dropping & creating the tables is done using the embedded queries in the Java application.

- **MobileDevice:**
  - This table stores all the unique MobileDevice's SHA-256 hash values. This deviceHash is inserted into the table which is a Primary Key, so no duplicate entries are made into this table. The records are inserted into this table when the current MobileDevice syncs with the Government, which in turn communicates with the database.
  - The synchronize () method is called by the MobileDevice class which in turn calls the mobileContact () method in the Government class.

- **TestResult:**
  - This table stores all the testHashes for the tests that any individual person took. The Test_Date along with the Test_Result of the test is stored in the table. The testHash is also a SHA-256 hash value, which is the primary key in the table. The records are inserted into the table when recordTestResult () method is called by the Government. The Test_Result is a boolean value either '**true'** or '**false'**.

- **MobileDevice_TestResult:**
  - This table is an associative table between the MobileDevice & TestResult tables. It stores the primary keys of both the tables as foreign keys also forming composite primary key of the table. Data is inserted into the table when the synchronize () method is invoked after calling the positiveTest () method. This table has foreign keys from MobileDevice & TestResult tables, which makes it compulsory to insert in either of the tables first. So, it is mandatory to invoke the recordTestResult () method before the positiveTest () followed by synchronize () method.

- **Contact:**
  - This table stores the contact details of the individuals, for a particular Contcat_Date, Contact_Duration. There is also a Contact_Notified column that stores boolean '**true'** or '**false'** for whether the contact has been notified or not. This table has an auto_increment key ContactID as the primary key. While recording any new contact the Contact_Notified is set to '**false'** by default. It is updated to '**true'** if the device has come in contact with someone directly who tested positive for COVID – 19 in a 14-day period. The ContactID's of people who have Contact_Notified as **'false'** & have come in contact with a COVID – 19 positives in a 14-day period are retrieved from the database, & their Contact_Notified column is updated to '**true'**, meaning they have been notified.

The ER diagram for the database design can be referred in the below figure:

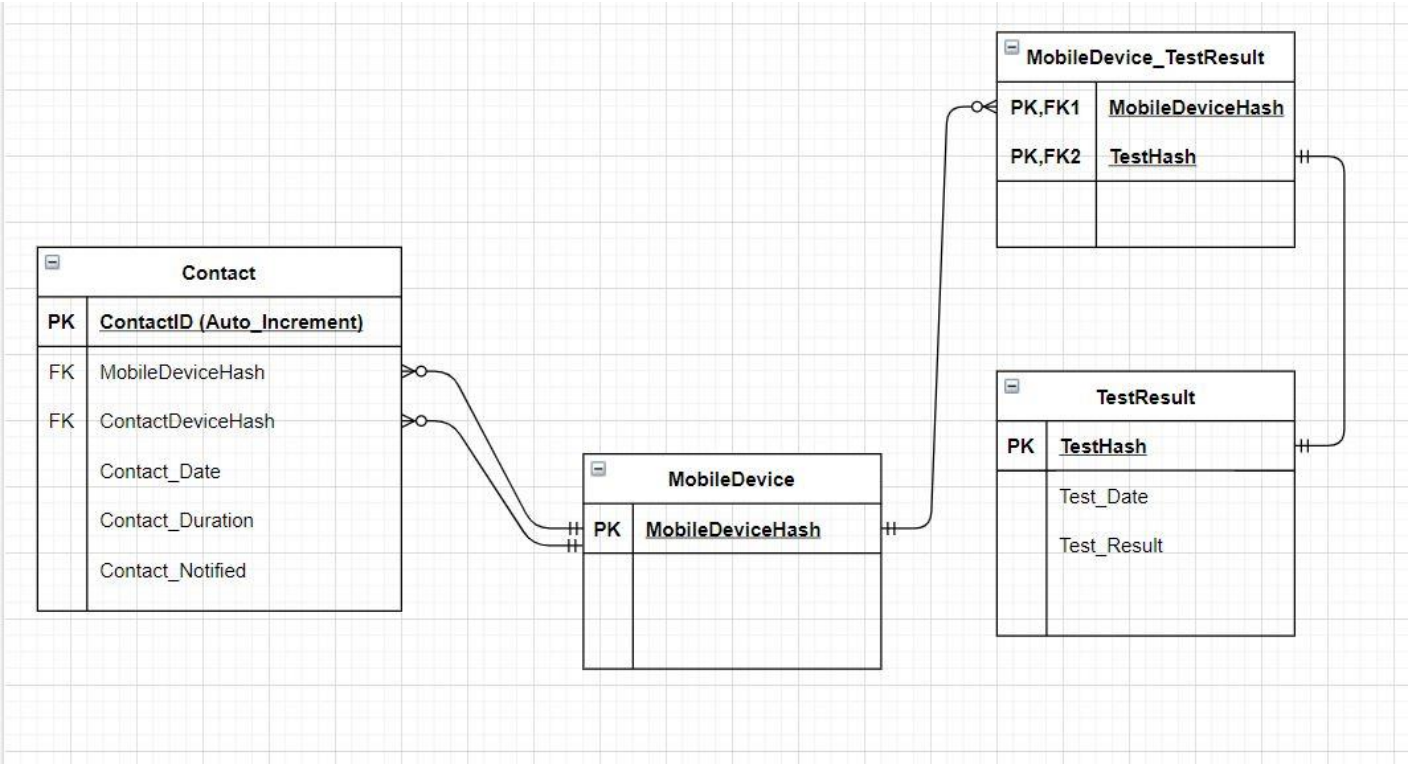

Figure 1: Database Deign

Implementation Details:

1. **MobileDevice class:**
   This class stores the config details of the mobile device (address & deviceName) as a SHA-256 hash value, which is unique for each device. The mobile device's details are not stored anywhere only the hash value is stored & passed to the Government class. This class records all the contacts the device has with any other device & also records all the positive tests the device has taken. All this data is sent to the Government class via the synchronize method.

   a. **MobileDevice (String configFile, Government contactTracer)**
      i. This parameterized constructor for MobileDevice class accepts the config file containing address and deviceName, converts them into SHA-256 hash which will uniquely identify that mobile device. The device's details are not stored anywhere & the hash value is also stored in a private global variable which can be accessed through **getHash ()** public method.
      ii. The constructor first validates the correctness of input and throws a RuntimeException with a display message of what went wrong. If all inputs are correct, it uses an instance of Properties class to read data from the config file.

      Note: Multiple instances can be created with different config files, if the contents of the config file are same then the hash value will also be same.

   b. **recordContact (String individual, int date, int duration)**
      i. The recordContact () method checks for input validations, throws RuntimeException if the input parameters passed are invalid, converts the date (number of days since 1st

Jan 2021) passed into yyyy-mm-dd format using a method getLocalDate () which uses LocalDate class and stores it in the global arraylist of Contact objects.

ii. This Contact object stores contacted individual, date and duration.

Note: To record contacts between devices, the deviceHash can be used directly by calling the getHash () method with the MobileDevice's instance.

c. positiveTest (String testHash)

i. The positiveTest () method checks for input validations, throws RuntimeException if the input parameters passed are invalid.

ii. It converts the passed string into a SHA-256 hash value, and appending the passed testHash string at the end of the generated hash (just for reference).

iii. It also stores this in the instance of the mobile device and stores test hashes of recorded positive tests as string in an arraylist.

d. synchronizeData ()

i. The synchronizeData () method checks for data-flow test cases, either retuen true or false, builds an XML file using DOM Parser class

1. It includes all the contacts that the current device had & test hashes in the XML format
2. This XML document is then formatted into a string object named **contactInfo** using StringBuilder class.
3. The method then invokes the **mobileContact ()** method using **contactTracer** object of the Government class.
4. If the mobileContact () method returns true (meaning the current device has come in contact of someone positive in span of 14 days), then it clears all the data that was synced and returns true, else returns false.
5. This method also clears the contactTracer's testList & contactList that were use din the Government class as the data has been written to the database.

2. Contacts class:

The Contact class stores the contacted individual as string in the instance of the class, the contact date in the Local Date format **yyyy-mm-dd**, and the contact duration as int in the instance of the class. There is a parameterized constructor which assigns the values passed during instantiation of the class & stores it in the global variables defined. The main purpose of the class is to store the contact details of the individual device in an arraylist of Contact objects.

3. Government class:

This class is designed using Singleton Design Pattern, allowing the class to only have one instance at any point of time. This class stores the configuration details read from the config file, to connect to the database. This class reads the contactInfo string of XML format for the initiator passed. This class also uploads the data to the database using SQL queries & also query the database using select queries. It also has a method to find the gatherings at any given date.

a. Government (String configFile)

i. This is a parameterized constructor for the Government class that accepts the configuration file containing database connection details and stores them into global variables.
ii. The file details & file validation is done & if any exception faced then throws a RuntimeException.
iii. If all inputs of the file are validated correctly, it uses an instance of Properties class to read data from the file.
iv. Stores the database, user and password of the database in the global variable which will be used further in the program to establish connection to the database.

b. mobileContact (String initiator, String contactInfo) method:

i. The mobileContact () method is called from the synchronize () method of the MobileDevice class, passing the initiator string & contactInfo string in XML format.
ii. This method validates the input parameters passed, returns false if so.
iii. This method adds the initiator string to the MobileDevice table in the database if it does not exist, reads the data by passing the contactInfo XML formatted string input to readXML () method.
iv. It also uploads the data into the database after it read the contactInfo string successfully.
v. Then it executes a select query that will retrieve the contacts of the initiator that were positive (if any) and an update query that will update the Contact_Notified field of the Contact table in the database for those contacts that were reported to be exposed to a COVID positive.

Note: The SQL queries to insert the data, retrieve the data & update the data are embedded into this method.

c. recordTestResult (String testHash, int date, boolean result) method:

i. The recordTestResult () method validates if the input parameters are correct or else throws RuntimeException.
ii. It converts the passed testHash string into a SHA-256 hash value using Message Digest and stores it into a string.
iii. It also converts the date passed into Local Date format (yyyy-mm-dd)
iv. It connects to the database to insert data in the TestResult table in the database.

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

i. This method first validates all the inputs & if anyone of them fail then it throws a RuntimeException.
ii. Here the date is passed as number of days since 1st Jan 2021, so I am using getLocalDate () method to get the date in yyyy-mm-dd format using the Local Date class.
iii. This method basically executes a select query to fetch the contact data from the Contact table in the database, filtering based on the date passed.
iv. The data retrieved is stored initially in an arraylist, both in normal & reverse order.

v. Then from these arraylists the data is added into the main arraylist of arraylist allIndividualPairs (the normal pair is added as we check for both the pairs normal & reverse).

vi. Then I iterate through the allIndividualPairs.

1. So the logic behind my findGatherings is that I am considering a pair of individuals, then I using them I iterate through the entire pairs list, I search for contact that both the individuals in the above pair had.
2. If found the I add the new individual as well as their respective contacts in the gathering.
3. I also look for reverse order pairs or duplicates as they shouldn't be considered.
4. The query that I am using basically adds the same contacts duration together & passed me the grouped contact between the individuals with duration being summed.
5. Then now I iterate using another pair through the allIndividualPairs list, I have written a contactSearch () method which is a private method.
    a. This method basically searches for any contacts the current pair has in the main list.
6. I keep adding the contacts that qualify to be in the gathering, as there is a filter of minSize. The gathering should be greater than the minSize then only it can be considered as a gathering.
7. There is also a minTime constraint that I am checking while retrieving the data from database.
8. Then after finalizing the pairs in the gathering, there is a formula to calculate the density
   n = number of individuals in the gathering
   c = number of pairs of individuals in the gathering
   m = n* (n-1)/2
   d >= density, which is passed then only this can be considered as a gathering

Note: I am not removing the gatherings that are reported once, if the government calls findGatherings () twice it will get the same result. I have considered it this way because whenever government calls findGatherings () it should get the gatherings for that date.

Example for my logic:

A B

A C

A D

B C

C D

D E

Gathering: (A,B), (A,C), (B,C), (A,D), (C,D) -> 1 gathering found

Individuals: A B C D

Test Cases for Final Project:

Tried to cover all possible test cases here as well as in the JUnit test plan.

1. **Input Validation Test Cases:**
    a. **MobileDevice Class**

        i. **public MobileDevice (String configFile, Government contactTracrer)**
            1. Empty string passed as configFile
                a. Throws RuntimeException
            2. Null value passed as configFile
                a. Throws RuntimeException
            3. String with multiple spaces passed as configFile
                a. Throws RuntimeException
            4. configFile passed exists
                a. Accepts the input & generates the deviceHash
            5. configFile passed does not exist
                a. Throws RuntimeException
            6. Contents of the configFile passed are valid
                a. Accepts the input & generates the deviceHash
            7. configFile does not contain the address field
                a. Throws RuntimeException
            8. configFile does not contain the deviceName field
                a. Throws RuntimeException
            9. configFile address filed is empty/null
                a. Throws RuntimeException
            10. configFile deviceName is empty/null
                a. Throws RuntimeException
            11. contactTracer is null
                a. Throws RuntimeException
            12. contactTracer is a valid Government class object
                a. Accepts input & generates the deviceHash
            13. configFile extension is a **.properties/.txt/without any extension**
                a. Accepts the input & generates the deviceHash

        ii. **public void recordContact (String individual, int date, int duration)**
            1. String individual passed is null/empty
                a. Throws RuntimeException
            2. String individual passed contains multiple spaces
                a. Throws RuntimeException
            3. Date passed is a positive integer

a. Accepts the input

4. Duration passed is a positive integer
    a. Accepts input

iii. public void positiveTest (String testHash)

1. testHash string passed is null/empty
    a. Throws RuntimeException
2. testHash string passed with multiple spaces
    a. Throws RuntimeException
3. Same testHash string passed multiple times
    a. Accepts the input

iv. public boolean synchronizeData ()

1. No input validation tests for this method as it takes all the data from the MobileDevice's instance.

b. Government Class

i. Private Government (String configFile)

Note: I have kept the constructor private as I am implementing Singleton Design Pattern for Government class. Detailed explanation is given in the implementation details & assumptions section.

1. Empty string passed as configFile
    a. Throws RuntimeException
2. Null value passed as configFile
    a. Throws RuntimeException
3. String with multiple spaces passed as configFile
    a. Throws RuntimeException
4. configFile passed exists
    a. Accepts the input & stores the database connection values
5. configFile passed does not exist
    a. Throws RuntimeException
6. configFile extension is a **.properties/.txt/without any extension**
    a. Accepts the input & stores the database connection values
7. Contents of the configFile passed are valid
    a. Accepts the input & stores the database connection values
8. configFile does not contain the database field
    a. Throws RuntimeException
9. configFile does not contain the user field
    a. Throws RuntimeException
10. configFile does not contain the password field
    a. Throws RuntimeException
11. configFile database filed is empty/null
    a. Throws RuntimeException
12. configFile user is empty/null
    a. Throws RuntimeException
13. configFile password is empty/null
    a. Throws RuntimeException

ii. public boolean mobileContact (String initiator, String contactInfo)

<u>Note:</u> This method is ideally not required to be tested for Input Validation tests as all the input comes from the Synchronize () method of the MobileDevice class.

1. Null/empty passed as initiator
    a. Throws RuntimeException
2. Null/empty passed as contactInfo
    a. Throws RuntimeException
3. Short string passed as contactInfo
    a. Accepts the input
4. Long string passed as contactInfo
    a. Accepts the input

iii. public void recordTestResult (String testHash, int date, boolean result)

1. testHash string passed is null/empty
    a. Throws RuntimeException
2. testHash string passed with multiple spaces
    a. Throws RuntimeException
3. Same testHash string passed multiple times
    a. Accepts the input
4. date passed in negative
    a. Throws RuntimeException
5. date passed is a positive integer
    a. Accepts the input
6. result passed as boolean true/false
    a. Accepts the input
7. result passed anything other than boolean value
    a. Throws RuntimeException

iv. public int findGatherings (int date, int minSize, int minTime, float density)

1. date passed in negative
    a. Throws RuntimeException
2. date passed is a positive integer
    a. Accepts the input
3. minSize passed as a positive integer
    a. Accepts the input
4. minSize is passed as a negative integer
    a. throws RuntimeException
5. minTime passed as a positive integer
    a. Accepts input
6. minTime passed as a negative integer
    a. Accepts the input

2. Boundary Test Cases:

a. MobileDevice Class

i. **public MobileDevice (String configFile, Government contactTracrer)**
   1. configFile name is a very long
      a. Accepts the input & generates the deviceHash
   2. configFile name is very short
      a. Accepts the input & generates the deviceHash
   3. configFile extension anything other than **.properties/.txt/without any extension**
      a. throws RuntimeException

ii. **public void recordContact (String individual, int date, int duration)**
   1. String individual passed is very long
      a. Accepts the input
   2. String individual passed is very short
      a. Accepts the input
   3. Date passed is 0
      a. Accepts the input
   4. Date passed in negative
      a. Throws RuntimeException
   5. Duration passed is negative
      a. Throws RuntimeException
   6. Duration passed is 0
      a. Throws RuntimeException

iii. **public void positiveTest (String testHash)**
   1. Short string passed
      a. Accepts the input
   2. Very long string passed
      a. Accepts the input

iv. **public boolean synchronizeData ()**
   1. No boundary tests for this method as it takes all the data from the MobileDevice's instance.

b. Government Class

i. **private static Government getInstance (String fileName)**
   1. configFile name is a very long
      a. Accepts the input & stores the database connection values
   2. configFile name is very short
      a. Accepts the input & stores the database connection values

ii. **public boolean mobileContact (String initiator, String contactInfo)**
   1. Short string passed as initiator
      a. Accepts the input
   2. Very long string passed as initiator
      a. Accepts the input

iii. public void recordTestResult (String testHash, int date, boolean result)

1. date passed is 0
    a. Accepts the input
2. Short string passed
    a. Accepts the input
3. Very long string passed
    a. Accepts the input

iv. public int findGatherings (int date, int minSize, int minTime, float density)

1. minTime passed as 1
    a. Accepts the input
2. density passed as 0
    a. Accepts the input
3. density passed as 1
    a. Accepts the input
4. density passed as a negative float value
    a. Throws RuntimeException
5. density passed as 2
    a. Throws RuntimeException
6. density passed anything greater than 1
    a. Throws RuntimeException
7. minSize passed as 1
    a. Accepts the input
8. minSize passed as 0
    a. Throws RuntimeException
9. minTime passed as 0
    a. Throws RuntimeException
10. date passed is 0
    a. Accepts the input

3. **Control Flow Test Cases:** I have covered these tests in the JUnit test plan.

4. **Data Flow Test Cases:**
    - **a.** Call Government getInstance () method twice in a row
    - **b.** Call MobileDevice () method multiple times in a row
        - **i.** Creating multiple instances
    - **c.** Call recordContact () method multiple times in a row
    - **d.** Call positiveTest () method multiple times in a row
    - **e.** Call synchronize () method multiple times in a row
    - **f.** Call positiveTest () method before calling recordContact () method
    - **g.** Call recordContact () method before calling positiveTest () method
    - **h.** Call synchronize () method before calling positiveTest () method
    - **i.** Call synchronize () method before calling recordContact () method
    - **j.** Call Government getInstance () method after calling synchronize method
    - **k.** Call recordTestResult () method multiple times in a row
    - **l.** Call recordTestResult () method before calling synchronize () method
    - **m.** Call recordTestResult () method before calling positiveTest () method

n. Call recordTestResult () method before calling recordContact () method
o. Call findGatherings () method multiple times in a row
p. Call findGatherings () method before calling synchronize () method
q. Call findGatherings () method before calling recordContact () method
r. Call findGatherings () method before calling positiveTest () method
s. Call findGatherings () method before calling recordTestResult () method

Note: mobileContact () method is never explicitly called, it gets called inside the synchronize () method.