CSCI 3901 Final Project

Ananthi Thiagarajan Subashini
B00874391

Overview:

This application is designed for COVID-19 management system. The application notifies if a person has been in the vicinity of any COVID-19 infected individuals. The Government systems can also detect number of large gatherings happened on a given date.

Files and External Data:

Covid application contains two source packages (src and test) and also has a folder called 'testfiles' which contains the property and configuration files used in the Junit test files.

- Source folder:
  Package – com.covid.main
  Class files:
  - MobileDevice.java
  - Government.java
  - Contact.java
  - ApplicationValidationException.java
  - Utility.java
  - MainDriver.java
- Test Folder:
  Package – com.covid.main
  Class files:
  - AllTests.java
  - FindGatheringsTest.java
  - GovernmentConstructorTest.java
  - GovernmentRecordTest.java
  - MobileConstructorTest.java
  - MobilePositiveTest.java
  - MobileRecordContactTest.java
  - SynchronizeAndControlFlowTest.java
- Testfiles- property and configuration files used for various test cases for MobileDevice and Government classes.
- Finalproject.sql – SQL file that contain queries to create schema and tables that are used in the project. Schema name can be changed but the name should be changed in the property file passed in the Government constructor.

- Configuration file for MobileDevice –
  deviceName=
  networkAddress=
- Database configuration file for Government –
  hostURL=
  port=
  additionalHeaders=
  username=
  password=
  schema=

Data structures and their relation to each other

MobileDevice.java contains six methods.

- MobileDevice(String configurationFile, Government contactTracer)
  i) Default constructor is replaced by the parametrised constructor.
  ii) Constructor accepts the device configuration file's path and instance of Government class as parameters.
  iii) Throws ApplicatioinValidationException in case of input validations.

- recordContact(String individual, int date, int duration)
  i) This method is used to record the neighbour interactions for given date and specified duration.

- positiveTest(String testHash)
  i) This methods records the covid test hash taken by the individual in the database.

- Synchronize()
  i) MobileDevice instance synchronizes with the Government network during the synchronize() method. Recorded contacts and test hashes are pushed to government server.

- String getContactListSize() – returns the number of contacts recorded so far by the Mobile Device.
- String getHashValuefin() – returns the hash value of the Mobile device
  String getTestHash() – returns the test hash passed by the Mobile device

Government.java contains nine methods, out of which only three is public. mobileContact() method is made protected to give limited access to the method and other methods are kept private for internal method use.

- Government(String configFile)
    i) Default constructor is replaced by the parametrised constructor.
    ii) Constructor accepts the database configuration file's path and tests the SQL connection.
    iii) Throws ApplicatioinValidationException in case of input validations.

- protected boolean mobileContact(String initiator, String contactInfo)
    i) Methods records mobile devices interactions and also checks for any covid-19 interactions in last 14 days.

- int findGatherings(int date, int minSize, int minTime, float density)
    i) This method returns the number of large gatherings occurred on the given date

- recordTestResult(String hash)
    i) This method is used to stored the test data provided by the test centre.

- private createSQLConnection – Used to create SQL connection
- private checkDBProperties – checks if all the parameters are present in the property file
- checkInfected() and saveContactAndTestHash() are other two private methods used by the mobileContact() method.

ApplicationValidationException.java-

The application throws a validation exception in case of input validations and also handles and reframes other java exceptions.

Database structure:

1)Table – fp_mobile_dtls

fp_mobile_dtls
mobile_id
mobile_name
test_hash
Registerd_date

Column info:

- mobile_id – primary key
- mobile_name – unique
- test_hash – forign key (Table : fp_test_dtls, coloumn : test_hash)
- registered_date – Time of entry

2)Table- fp_device_log

fp_device_log
log_id
mobile_hash
neighbour_hash
date
duration
registered_date

Column info:

- log_id – primary key
- mobile_hash – foreign key (table : fp_mobile_dtls, coloumn: mobile_name)
- neighbour_hash - foreign key (table : fp_mobile_dtls, coloumn: mobile_name)
- date
- duration
- registered_date – Time of entry

3) Table – fp_test_dtls

fp_test_dtls
test_id
test_hash
test_date
test_result
registered_date

Column info:

- test_id – primary key
- test_hash – unique constraint
- test_date

- test_result
- registered_date – Time of entry

4) Table – fp_covid_interaction_dtls

fp_covid_interaction_dtls
interaction_id
log_id
Registerd_date

Column info:

- interaction_id – primary key
- log_id – forign key (Table : fp_device_log, coloumn : log_id)
- registered_date – Time of entry

Assumptions

- If the mobile device, fails to connect with the government during synchronize(), the contacts recorded should be saved to synchronize in the future.
- The date value passed in recordContact(), recordTestResult() and findGatherings() method should not be future date.
- Configuration file passed to device can either be text(.txt) or property(.properties) file or simply a plain file.
- Similarly, database file passed to government constructor can be text(.txt) or property(.properties) file or simply a plain file.
- The network address passed can either be ip address or mac address.
- In find gatherings method, two devices who had contacted each other will have two separate entries. Both of them would record the interaction in the database.
- The individual value passed in the recordContact() method is also a hash value combination (deviceName + networkAddress).
- Test hash provided by the test centre is unique for each test details.

Key Algorithm and design elements

MobileDevice(String configurationFile, Government contactTracer)

- Default constructor is replaced by the parametrised constructor.
- Constructor accepts the device configuration file's path and instance of Government class as parameters.
- The configuration file and government object should not be empty or null.
- Also, the deviceName and networkAddress should not be null.

- Hash value is created by concatenating the deviceName and networkAddress using default. String.hashCode() method.
- The hash value created is stored in a global variable and the government object is also stored in a global variable to be accessible by other methods.
- The values from the configuration file are fetched using **FileInputStream**(java.io.FileStream) and **Properties**(java.util.Properties) class
- Throws ApplicatioinValidationException in case of input validations.

recordContact(String individual, int date, int duration)

- This method is used to record the neighbour interactions for given date and specified duration.
- The individual hash value passed should not be null or empty.
- Date value should not be negative and also should be future date.
- Duration must be within 1-1440.
- ApplicationValidationException is thrown in case, the parameters does not meet these constraints.
- Valid neighbours are stored in a list as Contact objects. (**List<Contact>**)
- Values are saved in the list, instead of updating directly in the file because, in case synchronize() method fails to connect to database and after that two more contacts are recorded, all the new and old neighbours should be passed in the next synchronize(). File updation would be difficult, so list is used. Values can be cleared on successful synchronization or can be retained in other scenarios.

positiveTest(String testHash)

- This methods records the covid test hash taken by the individual in the database.
- Test hash is checked for empty and null conditions.
- The value is stored in a global variable and accessed during the synchronize() method.

Synchronize()

- MobileDevice instance synchronizes with the Government network during the synchronize() method. Recorded contacts and test hashes are pushed to government server.
- The neighbour details are stored as List<Contacts>. The list is iterated to frame the XML file.
- A **directory** called **Mobile_devices** is created and the XML files are created in the directory. The hash value of mobile is used as filename while creating the XML file.
- The file is created if it doesn't exist using file object createNewFile() method.
- **PrintWriter** object is used to empty the file contents if the file has any previous values.

- Each contact recorded is appended to the XML file using **Files.write**.

```
<mobile>
    <device>
        <hash>.....</hash>
        <date>.....</date>
        <time>.....</time>
    </device>
    <device>
        ……
    </device>
</mobile>
```

Positive test hash value is also appended into the XML file.

```
<positive-test>
    <test-hash>……. </test-hash>
</positive-test>
```

- mobileContact() method is called and hash value and file name is passed as arguments.
- Incase of any exceptions, the list is not emptied and the neighbour details are retained for future synchronize().

Government(String configFile)

- Default constructor is replaced by the parametrised constructor.
- Constructor accepts the database configuration file's path and tests the SQL connection.
- createSQLConnection() method is used to create SQL connection and checkDBProperties() methods checks if all the parameters are present in the property file.
- The values from the database.properties file are fetched using FileInputStream (java.io.FileStream) and Properties(java.util.Properties) class.
- Parameters stored in the property file:

```
hostURL=
port=
additionalHeaders=
username=
password=
schema
```

- Throws ApplicationValidationException in case of input validations.

recordTestResult(String hash)

- This method is used to stored the test data provided by the test centre.
- The hash value is checked for null and empty conditions.
- Incase, the hash value is already recorded the method will throw application validation exception.

protected boolean mobileContact(String initiator, String contactInfo)

- Methods records mobile devices interactions and also checks for any covid-19 interactions in last 14 days.
- The method calls the internal method saveContactAndTestHash() method to insert the contact details in the database.
- The device and neighbour devices are first registered in a separate table called fp_mobile_dtls.
- PreparedStatement is created to insert the contact details. XML file is iterated and the hash values, date and duration is added to prepared statement as addBatch() and executed at a single time using executeBatch().
- DocumentBuilderFactory, DocumentBuilder, NodeList, Node, Element packages are used to read from the XML file.
- Nodelist is iterated for each <device>…..</device> values and for each iteration, the contact is added to the prepared statement batch.
- Separate connection is created for updating the test hash of the mobile device in the database.
- Invalid test hashes are logged using Logger methods.
- **setAutoCommit** is set as **false** before inserting contact and test hash and are committed using **connection.commit()** methods. In case of any errors, **rollback**() operation is performed.
- checkInfected() is called to check any covid interactions in the last 14 days. Both test date and interaction date should be within last 14 days. The interactions are added to a separate table called as fp_covid_interaction_dtls if it is already not there. Statement.**executeUpdate**() is used which returns the **number of rows** affected. If the affected count is greater than 0, the method returns true.

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

- This method returns the number of large gatherings occurred on the given date.
- Date value is passed to the fp_device_log table and values are fetched from that table accordingly.

```
String sql=
"SELECT mobile_hash, GROUP_CONCAT( CONCAT(neighbour_hash,"=",duration
) )AS neighbour_hash FROM fp_device_log WHERE date="+date+" GROUP BY
mobile_hash";
```

- Results are grouped by mobile_hash
- GROUP_CONCAT is used to concat the neighbour_hash.
- Neighbour_hash and duration column are concatenated into single column with "="
- Result will be like this,

Mobile_hash	Neighbour_hash
A	B=30, C=15, D=10
B	A=30, E=6, F=12
C	A=15, D=12
D	A=10, C=12, E=12
E	B=6, D=12, F=9
F	B=12, E=9

- Result is stored in a Map of map as,
  Map< String, Map<String,Integer>>
  So, the map will typically look like,
  {
  A={B=30,C=15, D=10},
  B={ A=30, E=6, F=12},
  C={ A=15, D=12},
  D={A=10, C=12, E=12},
  E={ B=6, D=12, F=9},
  F={ B=12, E=9}
  }

- **ConcurrentHashMap** is used to avoid concurrent modification exceptions.
- The key values from the map are stored in a set.
- The set is then converted into an array and the array is iterated.
- The array iteration starts with x value (from 1st to last element) and y value ((x+1)th element to avoid considering same pair twice. Element in x and y position are taken as first and next element and their interactions are taken from map.
- A set is created with first element and next element along with their common interactions. If the set.size() is > minSize, number of connections that exceeds mintime is determined.
- Similar logic is used to determine the connections. Set is converted to array, the array is iterated considering two pairs from i to n and j=i+1 to n, where n is the set size.
- The value of the first element is fetched from the global map. And that value is again a map and from that map, the interaction time with the next pair element is taken.
- If the interaction time is less than minimum time, cross-checking is done.
- The interaction time recorded by the next element is also checked if it exceeds minimum time (In some cases, some devices may not be synchronized() properly).

- If density calculated is greater than the density passed, increasing the gathering count and removing the nodes in Set from the Set of available nodes to search.
- If a large gathering is deducted, the nodes are removed from the set and Array iteration will occur only for the nodes which are in the set.

Limitations

- If two devices are recorded continuously in a single day and the total duration exceeds more than 1440, the validation is not checked here. The application will only check for duration at single time, if it exceeds 1440, not the combined duration.
- There are some limitations in finding the gatheringsCount() in the findGatherings() method. Different approach can yield different results.
- No user component or Bluetooth hardware components are added. The users must manually run the application to save and view the results.

TEST PLAN

Input validation

1) Null value passed as configuration file in MobileDevice constructor.
2) Empty value passed as configuration file in MobileDevice constructor.
3) Null value passed as Government object in MobileDevice constructor.
4) Null value passed as device name in MobileDevice constructor.
5) Null value passed as network address in MobileDevice constructor.
6) Empty value passed as device name in MobileDevice constructor.
7) Empty value passed as network address in MobileDevice constructor.
8) White spaces passed as device name in MobileDevice constructor.
9) White spaces passed as network address in MobileDevice constructor.
10) Null value passed as individual in recordContact() method.
11) Empty value passed as individual in recordContact() method.
12) White spaces passed as individual in recordContact() method.
13) Null value passed as test hash in positiveTest() method.
14) Empty value passed as test hash in positiveTest () method.
15) White spaces passed as test hash in positiveTest () method.
16) Null value passed as database configuration file in Government constructor.
17) Empty value passed as database configuration file in Government constructor.
18) Null value passed as test hash in recordTestResult().
19) Empty value passed as test hash in recordTestResult().

Boundary cases

1) Incorrect file path- configuration file doesn't exist in MobileDevice constructor.
2) Negative value passed for date in recordContact().
3) Future date value passed for date in recordContact().
4) Negative value passed for duration in recordContact()
5) Zero passed for duration in recordContact().
6) Values greater than 1440 passed for duration in recordContact().
7) Call synchronize() when database is down.
8) Call synchronize() with invalid database details.
9) Call synchronize() when schema has no tables.
10) Restart DB, and call synchronize() if previous values are still added.
11) White spaces passed as database configuration file in Government constructor.
12) Invalid file name passed for database configuration file in Government constructor.
13) Parameters missing in database configuration file in Government constructor.
14) Additional arguments added in database configuration file in Government constructor.
15) Schema doesn't exist in database configuration file in Government constructor.
16) Invalid database credentials in Government constructor.
17) White spaces value passed as test hash in recordTestResult().
18) Test date value > current date.
19) Negative value passed as date values.
20) Find gatherings – date value > current date
21) Find gatherings – date value < 0
22) Find gatherings – Minimum time < 0
23) Find gatherings – Minimum time = 0
24) Find gatherings – Minimum time > 1440
25) Find gatherings – Minimum size < 0
26) Find gatherings – Density < 0

Control flow tests

1) Invalid government object passed in Mobile device constructor.
2) Check the contact list size after adding values in recordContact().
3) Check the contact list size after synchronizing with the mobile network.
4) Check the contact list size after synchronizing and adding values in recordContact().
5) Invalid test hash passed.
6) Check if the Mobile_devices directory is created.
7) Check if the XML file is created properly.
8) Call government constructor when SQL server is down.
9) Create multiple mobile devices with the same government object and call synchronize().
10) Already exisiting test value passed as test hash in recordTestResult().
11) Find gatherings - Passing devices, some of which may haven't synchronized to government properly.

12) Find gatherings - Passing devices, in which one device has entry of other and the other device has no record of this device.

Data flow test cases

1) Correct file path with valid network address and device name in Mobile device constructor.
2) Check the contact list size after synchronize() throws an exception.
3) Check the contact list size after synchronizing throws an exception and again adding values in recordContact().
4) Updating test hash for same mobile device.
5) Call synchronize() with no recorded device.
6) Call synchronize() with one recorded device.
7) Call synchronize() with no positive test hash.
8) Call synchronize() with positive test hash.
9) Call government constructor with empty arguments in the property files.
10) Get gathering count for valid conditions.
11) Create mobile devices with same government object, synchronize() multiple times and in the middle update the test hash(positive) and checks if the synchronize() returns true. Checks if the synchronize() returns false again
12) Check if same covid interactions are being considered.

SQL Scripts

```
CREATE DATABASE IF NOT EXISTS finalproject_covid;

USE finalproject_covid;

DROP TABLE IF EXISTS `fp_covid_interaction_dtls`;
DROP TABLE IF EXISTS `fp_device_log`;
DROP TABLE IF EXISTS `fp_mobile_dtls`;
DROP TABLE IF EXISTS `fp_test_dtls`;

CREATE TABLE `fp_test_dtls`
(
 `test_id` INT(10) NOT NULL AUTO_INCREMENT,
 `test_date` INT(10) NOT NULL,
 `test_hash` VARCHAR(20) NOT NULL,
 `test_result` VARCHAR(20) NOT NULL,
 `registered_date` DATE,
 PRIMARY KEY (`test_id`),
        UNIQUE(test_hash)
```

```
);


CREATE TABLE `fp_mobile_dtls`
(
`mobile_id` INT(10) NOT NULL AUTO_INCREMENT,
`mobile_name` VARCHAR(20) NOT NULL,
`test_hash` VARCHAR(20) ,
`registered_date` DATE,
PRIMARY KEY (`mobile_id`) ,
UNIQUE(mobile_name) ,
CONSTRAINT `fp_test_hash` FOREIGN KEY (`test_hash`) REFERENCES `fp_test_dtls`(`test_hash`)
)
;



CREATE TABLE `fp_device_log`
(
 `log_id` INT(10) NOT NULL AUTO_INCREMENT,
 `mobile_hash` VARCHAR(20) NOT NULL,
 `neighbour_hash` VARCHAR(20) NOT NULL,
 `date` INT(10) NOT NULL,
 `duration` INT(10) NOT NULL,
 `registered_date` DATE,
 PRIMARY KEY (`log_id`) ,
 CONSTRAINT `fp_mobile_hash` FOREIGN KEY (`mobile_hash`) REFERENCES
`fp_mobile_dtls`(`mobile_name`),
 CONSTRAINT `fp_neighbour_hash` FOREIGN KEY (`neighbour_hash`) REFERENCES
`fp_mobile_dtls`(`mobile_name`)
 );


CREATE TABLE `fp_covid_interaction_dtls`
( `interaction_id` INT(10) NOT NULL AUTO_INCREMENT,
`log_id` INT(10) NOT NULL,
`registered_date` DATE,
PRIMARY KEY (`interaction_id`) ,
 CONSTRAINT `fp_log_dtls` FOREIGN KEY (`log_id`) REFERENCES `fp_device_log`(`log_id`)
 );
```