CSCI 3901 Winter 2021

Project

Due date: 23:59 AST Friday, April 16th, 2021 in Gitlab

Goal

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

Project Structure

The final project will have you apply your problem solving and development skills. The solution will require you to bring together

- Abstract data types and data structures
- Java implementation
- Basic algorithms
- Software development techniques including version control, testing, debugging, and defensive programming
- Good software program design
- Database design and use

While we provide a recommended problem for the course, you have the option of proposing a different problem for the project. A project proposal must include a non-trivial example of all the concepts mentioned above.

The project is **not** expected to include a user interface component or software that directly accesses a device's hardware.

Recommended Problem

One of the tools to help us manage the COVID-19 pandemic is contact tracing. When one person is diagnosed with COVID-19, the ability to notify other individuals who have been in contact with the person who is COVID-positive allows us to limit the spread of the disease faster.

At the same time, the ability to use contact information to detect the frequency of large gatherings also helps us understand the community's compliance with physical distancing advisories.

For this project, you will replicate scaled-down functionality of the Canadian federal government's application for contact tracing.

The big picture of the Canadian application

At a high level, each individual has an application on their mobile phones that detect instances of the same application on nearby mobile phones through Bluetooth connections. The phones record the date and duration of the contacts with other phones and periodically report these contacts, in an anonymized manner, to a central database in the government.

When an individual is diagnosed with COVID-19, that diagnosis is also recorded in the central database. Any individual who has been in contact with the diagnosed case within 14 days is then notified of the contact on the next time that their mobile phone contacts the central database.

We can also use the central database to check for large gatherings. Given a large gathering, many of the individuals at the gathering will report contact with the same or similar set of individuals. Identifying these clusters of tightly-connected individuals on a daily basis gives us a sign of how frequently large groups are gathering; we can then watch for increases in COVID-19 diagnoses following those events.

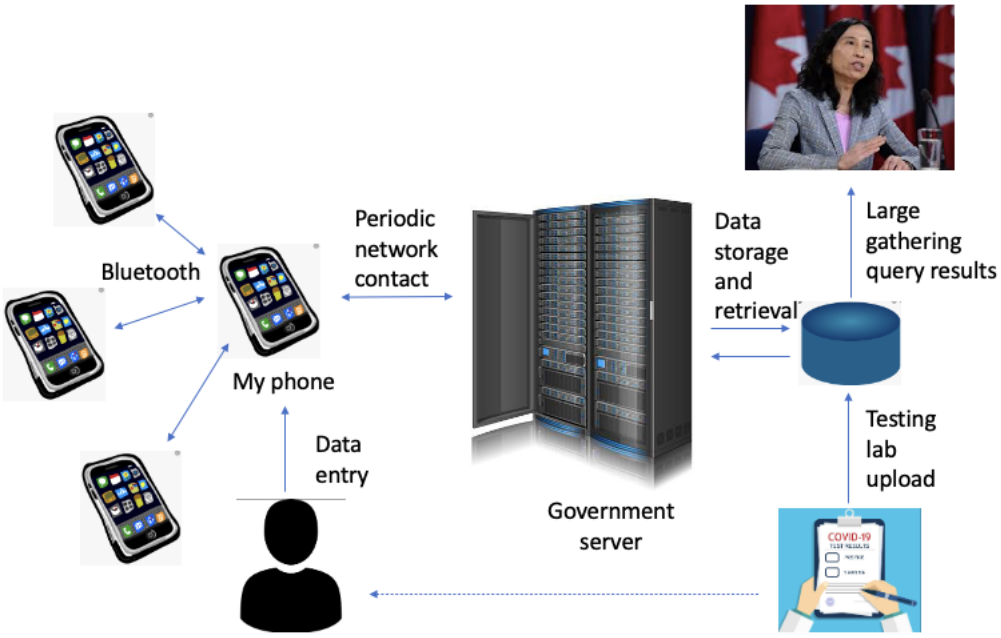

Figure 1: Canadian COVID-19 tracking application overview

Privacy is maintained in the system by having each individual identified by a hash of device information rather than with a login. The hash cannot be inverted. Mobile phones only exchange their hashes when they detect contact, and the mobile phones only report hashes (theirs and of others) to the Canadian government, so the government cannot identify who is who.

Project simplification

In this project, you will **not**

- Detect or create Bluetooth connections between devices
- Draw device information from the hardware to create a hash
- Create timers to periodically upload contact information from your implementation to the Canadian government's database (or a class database)

Instead, you will have a method to record a contact with someone else and a method to trigger an upload of your collected contact information to a database.

You will need to design at least two classes. The first class, called `MobileDevice`, will have the functions that the mobile phone would do:

- store contacts with others and upload the contacts to the central database,
- report that you have been tested positive for COVID-19 to the central database, and
- report back if the central database tells us that someone we've been around has been diagnosed with COVID-19.

The second class, called `Government`, will have the functions that are centralized:

- storing the overall set of contacts,
- storing the test results,
- notifying individuals who contact the database if they have been in contact with someone who has tested positive for COVID-19, and
- being able to report the number of large gatherings on any particular date.

The reporting of data from `MobileDevice` to `Government` will happen with method calls. In a full-scale implementation, they would communicate with one another through a network socket in a client-server architecture, but that's a topic for another day.

Specific methods are described in the next section, along with the criteria on what it means to detect a large gathering. You are permitted to add exceptions to these methods and return values... **as long as you document them**.

In the final solution, you will also design the database. Calls to the `MobileDevice` and the `Government` classes will happen from a `main()` method that you provide. Marking may also have JUnit tests that invoke the methods directly.

Functionality for `MobileDevice`

`MobileDevice(String configFile, Government contactTracer` Constructor for the class. It accepts a configuration file that contains the device's network address (string) and device name (string); these would normally come from the hardware, but we'll put them in the configuration file instead for simplicity. The configuration file will have one line for each configuration value.

The format of a line is `<keyword>=<value>` where `keyword` is either `address` or `deviceName`. You are welcome to add other configuration parameters.

`contactTracer` is the object that represents the government and that you will use to record your results.

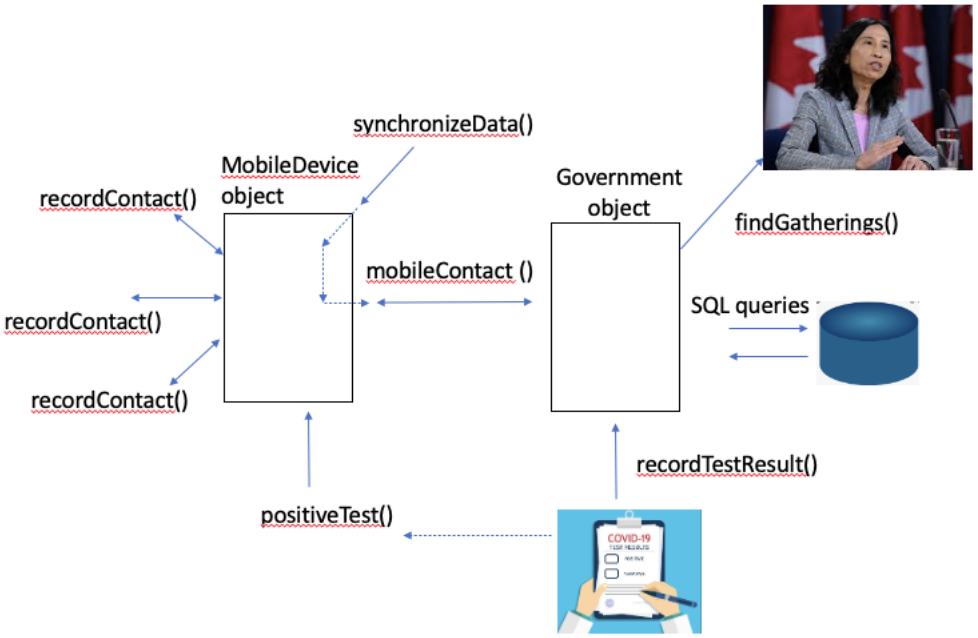

Figure 2: General interaction structure for `MobileDevice` and `Government` classes

`recordContact(String individual, int date, int duration)` When our Bluetooth driver (**not part of this project**) detects another device, that driver calls `recordContact` to locally record that `individual` (an alphanumeric string) was near us on `date` for `duration` minutes. The date is the number of days since January 1, 2021. These contacts do **not** go to the government system right away. They are stored in `MobileDevice` until the time comes to send the bulk of contacts to the government.

`positiveTest(String testHash)` The user interface (not part of this project) will call `positiveTest` when the user asks to tell the system that they have tested positive for COVID-19. The testing agency will provide an alphanumeric string that identifies the test; the user will record that string to link those results to the individual who normally uses the mobile device. `testHash` will need to be matched with a test result in the Government system.

`boolean synchronizeData()` Some process on the mobile device (not part of this project) will invoke this method periodically to trigger an exchange with the government database. When invoked, the application will package all of the information in the mobile device into a **formatted string** and send that information to the government through the `Government` object's `mobileContact` method. The information packaged up includes the device's hash, any contacts, and positive test hashes reported by the owner of the phone (as user calls to `positiveTest`). The method gets back from the government an indicator of whether or not this mobile device has been near anyone diagnosed with COVID-19 in the last 14 days. That outcome is the return value of `synchronizeData`: `true` if the device has been near someone with covid-19 and `false` otherwise.

The `MobileDevice` class will **not** write SQL statements to the government database. You will encode your data for the government object in a `String`; the format is of your choice, but I recommend that you consider organizing your data in an XML format. The government object, acting as a server, will then parse the string from the mobile device and store or retrieve data from the database.

Functionality for `Government`

`Government(String configFile)` Constructor for the class. It accepts a configuration file that contains the domain name of the database for the government (string), the username to access the database (string) and the password to access the database (string). The configuration file will have one line for each configuration value. The format of a line is `<keyword>=<value>` where `keyword` is either `database`, `user` or `password`. You are welcome to add other configuration parameters.

`boolean mobileContact(String initiator, String contactInfo)` This method is called by the mobile device `synchronizeData` method to send the contact information to the government. The caller is identified in the system by `initiator`, which is a `String`-valued **hash** of the mobile device's network address and name. The purpose of hashing the device address and name is to allow a device to be uniquely identified without revealing any information about the user. For a cryptographically secure hash value, I would recommend using the SHA-256 hash function from the `MessageDigest` class in `java.security`.

All the contact information is contained in the `contactInfo` string. You, as designer, get to define the structure of `contactInfo`. **Be sure to document your format fully**, either internally or externally.

The method returns whether or not the contacting mobile device has been near someone who tested positive for COVID-19 within 14 days of their contact. Only new COVID-19 contacts are reported back. These contacts **must** be stored in the government database to survive restarts of the program.

`recordTestResult(String testHash, int date, boolean result)` Record in the database that a test, identified by the alphanumeric string `testHash`, had a collection taken up on `date` (the number of days since January 1, 2021), and came out positive (`result` = `true`) or negative (`result` = `false`).

`int findGatherings(int date, int minSize, int minTime, float density)` Looking at the contacts reported on `date`, return the number of large gatherings on that date. Again the date is the number of days since January 1, 2021.

To detect a gathering, consider all pairs of individuals A and B. Find the set S of people that both A and B contacted on that day (including A and B). We only consider S as a gathering if it contains at least `minSize` individuals.

To determine whether the gathering S is *large*, count the number of pairs of individuals c within S who contacted one another on the given date for **at least** `minTime` minutes. Note that if S contains n individuals then there can be at most $m = \frac{n(n-1)}{2}$ possible pairs. We then compare c with this maximum: if c/m is greater than `density` then the gathering is deemed large and worth reporting. In such an instance, we then remove all individuals in S from our continued search for large gatherings on this day.

Project process

A Gitlab repository has been provided to you to use for your project. You can locate it and `clone` it from `https://git.cs.dal.ca/courses/2021-winter/csci-3901/course-project/xxx.git` where you should replace `xxx` with your CS user ID. You are strongly encouraged to use your Git repository for version control throughout the development of your project.

You will `push` the final version of your repository to the remote repository on Gitlab before the deadline, at which point I will disable pushing updates to the repository.

The final commit in your repository should include:

- The Java source files for your project
- Any test harness or Java tests you wrote for testing
- The SQL script to create your database
- A `.pdf` report detailing the design of your solution as well as your test plan (what tests you will and/or did run to ensure proper operation)

Milestones

Only the final submission is graded. These milestones are provided only as a rough guideline for completion of the project. Feel free to ask the course instructor or TA for feedback on any of these milestones.

- March 5: Understanding the problem
- March 19: Design of the solution. data structures, algorithms, & test plan
- April 2nd: Progress has begun on implementation
- April 16th: Project due

Marking scheme

- Documentation (10%)
- Test plan (15%)
- Overall design and coding style (25%)
- Working implementation, including efficient processing (50%)