This project is divided into following files:

Student:

StudentGUIHandler.java : File that handles GUI of student process

StudentHandler.java : File that handles interaction of Student process with MQS

StudentRecordUpdate.java : Class that design message format for incoming messages to Advisor/Notification process

Notification:

NotificationGUIHnadler.java : File that handles GUI of notification process

NotificationHandler.java : File that handles interaction of Notification process with MQS

MQS:

SqlUtil.java: Util class to provide database helper methods

MqsGUIHandler.java : File that handles GUI of MQS process

MqsHandler.java : File that handles interaction of MQS process with other processes

MqsRequestHandler.java : File that handles specific process connection with MQS, one per connection

MqsRequest.java : Commong message format for MQS which other process can use to pass around information

Advisor:

AdvisorGUIHnadler.java : File that handles GUI of Advisor process

AdvisorHandler.java : File that handles interaction of Advisor process with MQS

Common:

Other utility classes that provide helper for common functionalities

### How to run this ####

First, MQS process needs to be started and after that any process can be started in any other order.

(In fact, MQS is also not required to be started first but in that cases there will be messages on other

processes' GUI about no connection with MQS.)

Services can be restarted in any order. Student can send request for advisor using its GUI. It needs to pass

Student name and Course name for which advisor's decision is required. Once advisor's decision is available,

it should be visible on both Advisor and Notification process GUI.

####### Design ######

\*\* Connection Desing \*\*

Student process has one client socket which connectes to MQS.

MQS process has one server socket where all other process' client socket connects to.

Notifcation has one client socket which connects to MQS server socket.

Advisor process has one client socket which connects to MQS server socket.

\*\* Message exhange & storage \*\*

Messages are exchanged and stored in a specific way and it is described as below.

All incoming messages from Student/Advisor processes to MQS are first persisted to database followed by

in-memory object creation. This means that even if MQS process dies before creating in memory

objects, it can still reconstruct them on restart from persisted records.

In-memory objects are created to make operations faster so that every lookup should not read database

when MQS process is working smoothly. They are stored in one LinkedList in MQS process.

On startup, MQS reads the database and if any previously unprocessed messages exists then it creates

in-memory objects for them before processing anything else.

\* Database Design \*

MQS stores all messaging in SQL database and below is table definition.

Table Name: MqsMessageDetails

Columns: student\_name (varchar), course\_name (varchar), advisor\_decision (varchar), message\_status (varchar)

Primary Key: (message\_status, student\_name, course\_name)

"message\_status" shows status of the message in point of time. (i.e. aat what processing level it is currently)

###### How MQS guarantees messages are not lost even if it crash/restarts ######

A) Student to MQS push request

MQS first creates DB record and then creates in memory object and append it to messaging queue.

This ensures that messages are always there even if MQS restarts/crash.

B) MQS to Advisor pull response

MQS first sends all in memory records that requires Advisor's decision to Advisor. If sending is

successful then and then only MQS updates its status. This ensures that messages

are delivered before its status gets updated.

C) MQS to Notification pull response

MQS first sends all in memory records that already have Advisor's decision to Notification

process. If sending is successful then and then only MQS updates its status.

This ensures that messages are delivered before its status gets updated.

D) Advisor to MQS push request

MQS first updates DB record and then in memory object. This ensures that messages

are always there even if MQS restarts/crash.

###### KEY FUNCTIONALITY #####

1. Processes can be started in any order without any issue.

2. If MQS restarts, other processes connects to it automatically without any manual intervention.

3. Student request is guaranteed to be processes if it reaches to MQS initially. System can endure process crash/restarts.

###### REFERENCES ######

SQL & Java:

https://alvinalexander.com/java/java-mysql-insert-example-preparedstatement

https://stackoverflow.com/questions/11909324/creating-a-database-table-if-it-does-not-exist-in-java-production-code-and-confi

Sockets and object exchange:

https://stackoverflow.com/questions/27736175/how-to-send-receive-objects-using-sockets-in-java