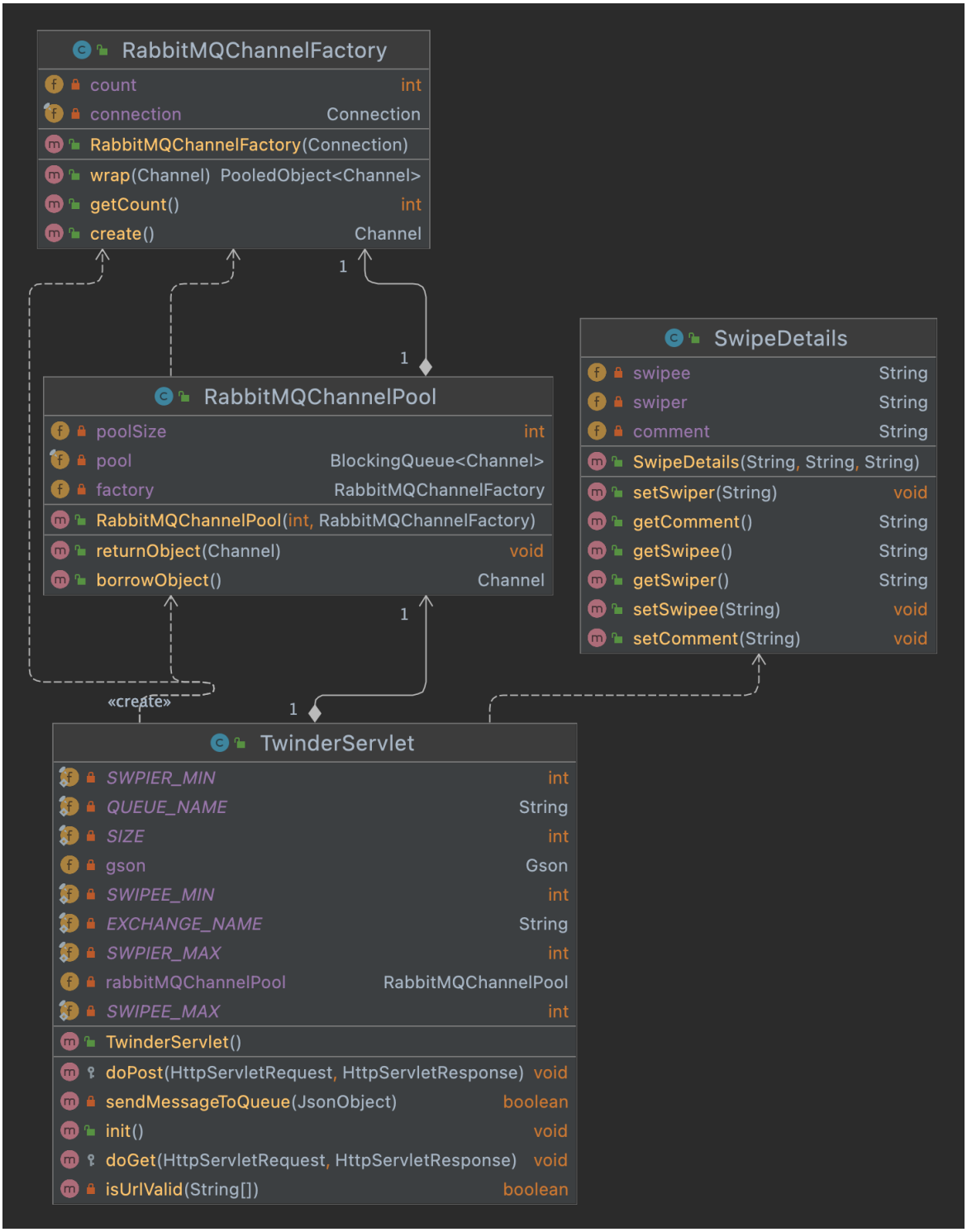
1. Siyuan Chen's GitHub Repo:

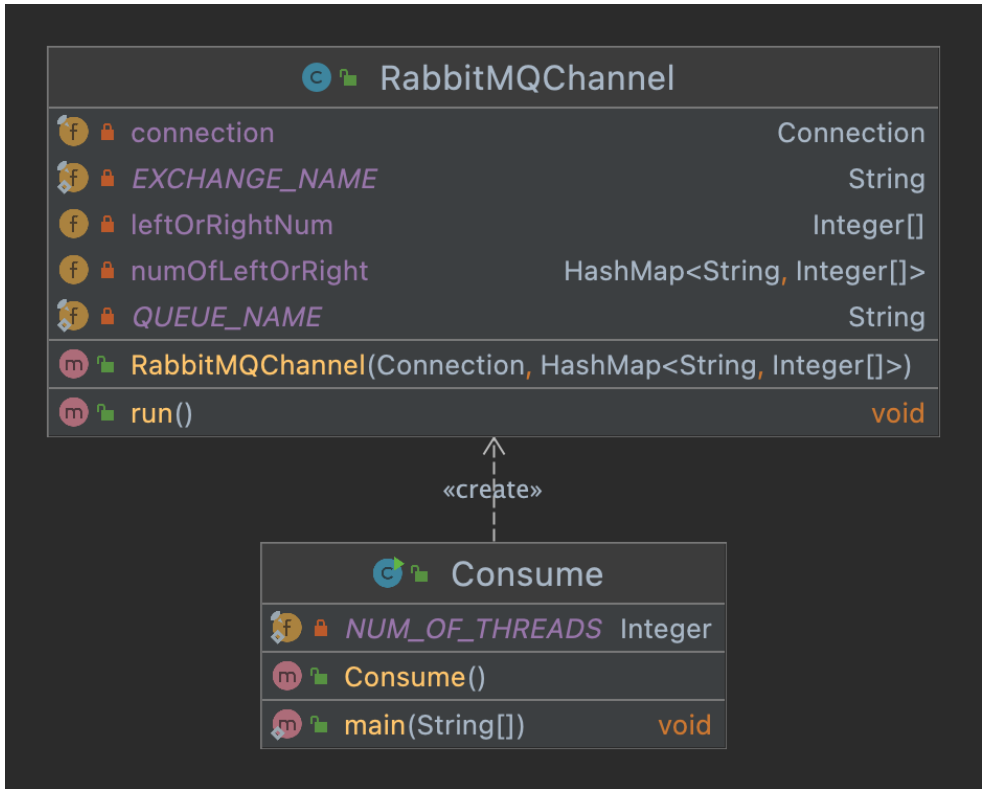
   https://github.com/Janice1457/Building-Scalable-Distributed-Systems/tree/main/HW2

Step 1 - Implement the Server-UML

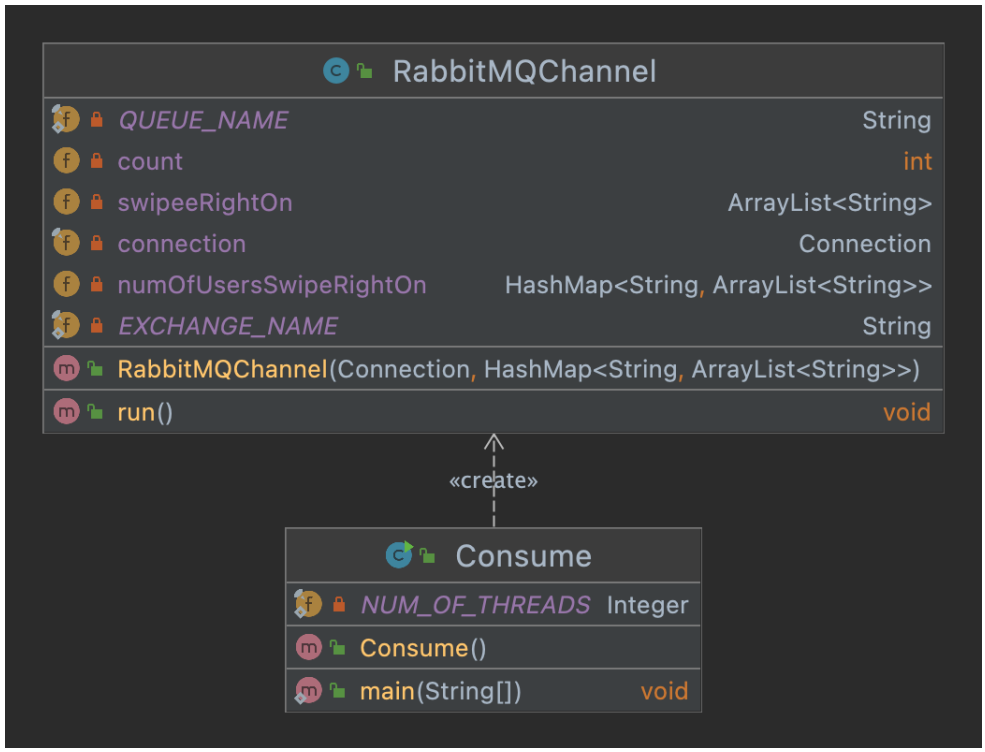


Step 2 - Implement Consumers-UML

Consumer1:



Consumer2:



2. Design:

Step 1 – Implement the Server:

I have read the textbook, and decided to use Producer and Consumer with a LinkedBlockingQueue pattern since This solution absolves the programmer from being concerned with the implementation of coordinating access to the shared buffer, and greatly simplifies the code.

There are four classes inside Servlet project, including SwipeDetails, RabbitMQChannelFactory, RabbitMQChannelPool, and TwinderServlet.

SwipeDetails: has 3 fields, including swiper, swipee, comment, and comment. These are response details for each POST request. For each POST my client needs to randomly generate values for the request. And In doPost method inside TwinderServlet class, it will parse the request and generate a swipeDetails object. Also it will instantiate a JsonObject swipeInfo with these swipe details properties.

RabbitMQChannelFactory: A simple RabbitMQ channel factory based on the APche pooling libraries. There are two fields, including connection and count. Connection is a valid RabbitMQ connection, and count is used to count created channels for debugging. And there is a create method which helps to create channels.

RabbitMQChannelPool: A simple RabbitMQ channel pool based on a BlockingQueue implementation. There are three fields, including pool, poolSize and factory. Pool is used to store and distribute channels. PoolSize is a fixed size of pool. Factory is used to create channels. And there are two methods, including borrowObject and returnObject. BorrowObject will help get a channel from the pool. ReturnObject can help return channel to the pool.

TwinderServlet: There are four main methods, including init, isUrlValid, doPost and sendMessageToQueue.

1) The init method can connect to a RabbitMQ node using the given parameters. Inside init method, I instantiate the connectionFactory and then set host, username, password and port. I also instantiate the connection rabbitMQChannelFactory and rabbitMQChannelPool to help connect to RabbitMQ.
2) The isUrlValid can validate if the urlPath is valid.
3) The sendMessageToQueue method can help get a channel from the pool, and publish message and return channel to the pool. I use fanout exchange type. I create a fanout exchange type and call it EXCHANGE, and it broadcasts all the messages it receives to all the queues it knows.

4) doPost method can help get the request and validate the swipeInfo and once sendMessageToQueue with valid swipeInfo returns true, then the response is successful. Otherwise, the request is not valid or failed to send message to RabbitMQ.

Step 2 – Implement Consumers

I use Consumer1 to support the first pattern: Given a user id, return the number of likes and dislikes this user has swiped.

And Consumer2 to support the second pattern: Given a user id, return a list of 100 users maximum who the user has swiped right on. These are potential matches.

Consumer 1 has a package called MyConsumer1 including RabbitMQChannel and Consume classes.

1) RabbitMQChannel class implements a runnable. There are two fields: connection and a hashmap data structure numOfLeftOrRight, the key is userId and the value is an integer array which has two integers. The first integer is total left number and the second integer is total right number. Once when leftOrRight is left then the first index number in the array get incremented, otherwise, the second index number in the array get incremented.

   Inside the run method, I use fanout exchange type. A consumer creates several anonymous queues in the broker, binds the queues to the exchange created by the publisher, and specifies that messages published should be delivered to this queue.

   And use a threadCallBack to get the message and translate into a JasonObject and then put messages into a hashmap, and consume that messages.

2) Consume class connects with RabbitMQ and also use multithread to run each thread concurrently.

Consumer 2 has a package called MyConsumer2 including RabbitMQChannel and Consume classes. And methods inside classes are similar to Consumer1.

1) RabbitMQChannel class implements a runnable. There are two fields: connection and a hashmap data structure numOfUsersSwipeRightOn, the key is userId and the value is an ArrayList swipeeRightOn which has 100 number of String swipee. If leftOrRight is right, then I add the swipee into swipeeRightOn arrayList. And Once the size of swipeeRightOn arrayList is more than 100, then I will use FIFO algo of a queue to remove the first swipee and add the new swipee at the end of the arrayList.

Inside the run method, I use fanout exchange type. A consumer creates several anonymous queues in the broker, binds the queues to the exchange created by the publisher, and specifies that messages published should be delivered to this queue.

And use a threadCallBack to get the message and translate into a JasonObject and then put messages into a hashmap, and consume that messages.

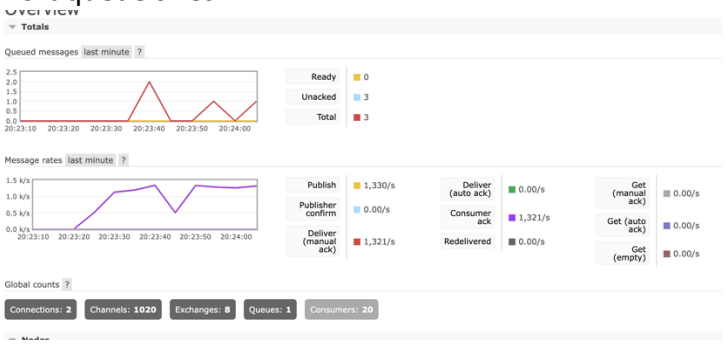
2) Consume class connects with RabbitMQ and also use multithread to run each thread concurrently.

3.Test Results:

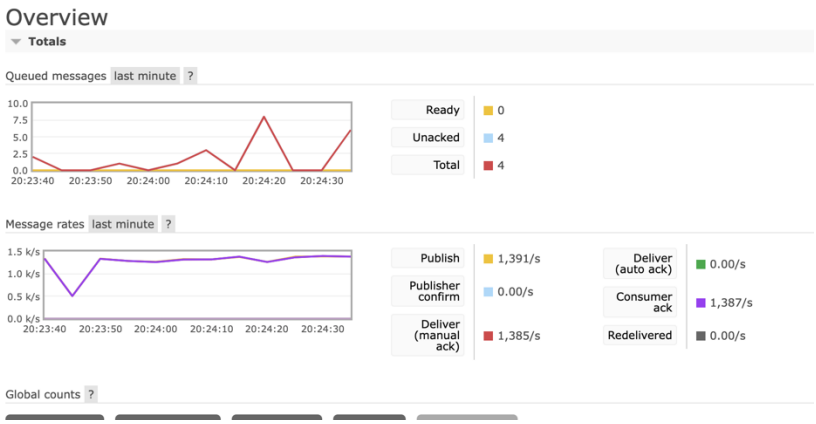
Single instance tests:

1) Overall throughput:

2) short queue sizes:

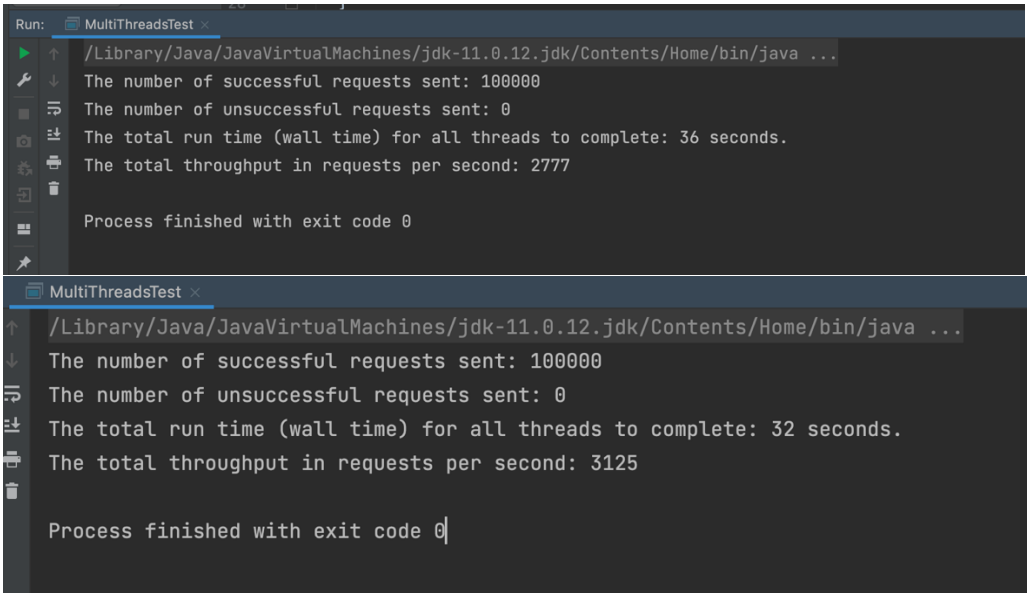


3) flat line profile:

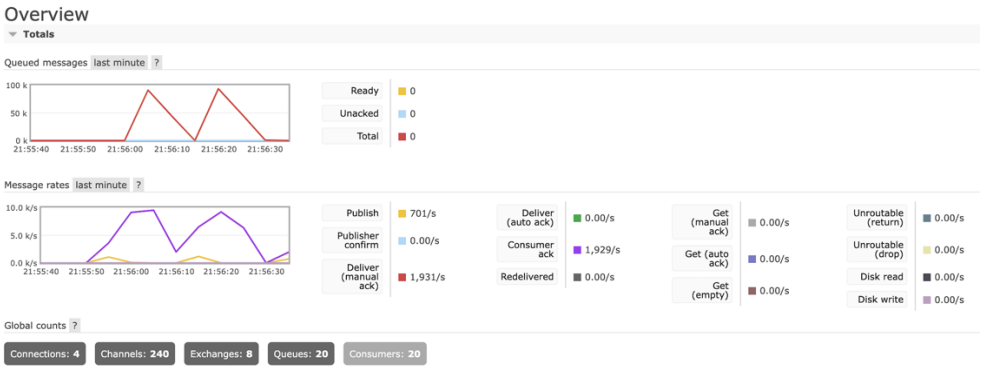


Load balanced instance tests:

1) Overall throughput:



2) short queue sizes:



3) flat line profile: