1. Simulate 3-Queue System [40 points]

We decided to create something more than a simple queue system to solve this problem. This program uses a series of block queue objects that pass a ‘person’ object to each in sequence. The resulting processes duration is then measured in real time using delay times that are generated using java random. This accomplishes several goals – to create a system that exhaustively uses queues, running multiple concurrent threads and simulate time based on system time to give a verifiable result of how each manipulated variable impacts the efficiency of the queue system.

A single delay queue was used to simulate the delay of each customer walking up to the security queue area. We decided to use the delay queue from concurrent as it seemed an organic usage. This worked by having a predetermined number of customers but each one only becoming available when they ‘arrived’. The process of which queue receives the customer is managed by the program itself as only one queue can take a person object from the arriving customer queue. Queue 1 and Queue 2 then respectively pass the person objects to subsequent blocking queues ‘toCheck1’ and ‘tocheck2’. From there, each feeds into ‘checker1’ and ‘checker2’. The person object is processed by each checker runnable. This process requires the checker thread to be able to ‘peek’ into the length of the queue only using a size method built into concurrent queues. We decided to use a ratio represented as a decimal percentage of how full the queue was with a default of 0.70 or about 70% full. When this condition is met the checker’s process time calculation is divided by two with each successive true result also speeding up the checker’s process time. Fine tuning of this would be required to give a full representation of a real checker’s behavior but for our purposes, we felt that it sufficed for a simulation represented by an amount of time for ‘processing’ only.

From ‘checker1’ and ‘checker2’, each passes to a single shared blocking queue that is then passed into the queue 3 runnable. From there, the behavior is the same except that single queue 3 thread feeds into another blocking queue ‘toDepart’. The blocking queue is then passed to a final checker runnable that processes and delays run time in similar fashion. For our purposes the program is completed at that point. The ‘checker3’ passes the processed ‘person’ objects to a final queue, ‘depart’. This could be used to extend the program further – perhaps into an entire airport simulation system.

The system uses milliseconds only in its measurement of time. We felt that this gave a more exact representation of computer time, allowing for greater precision in the understanding of results. If multiplied out or simply generated as a int/long number passed from one runnable to the next, it seemed like something might be lost in the accounting of the true processing time of each person object. By measuring start to finish in real time milliseconds, we get a representation of not only efficiency of each variable but a measurement of the run time of the overall program.

Challenges came at each stage of this program. Setting up each thread with each checker and queue runnable seemed to be the most difficult to understand but in the end, seemed to provide an explanation of the concurrent blocking queue being ‘locked’ by a respective thread. We solved this by creating a blocking queue for each stage, representing the links between each part of the system. The ‘arrive’ queue created a line of customers using a delay queue. The queue’s ‘toCheck1’ and ‘toCheck2’ each went to the checkers. From each checker we got a single queue ‘doneCheck’. This allowed a single shared queue to by used for the final queue 3 going to a single queue for the final check, ‘toDepart’. Upon exit, checker 3 gives us ‘depart’.

Once each runnable component was written, it was simply a matter of making all the pieces work together. In the end, it seemed that there are near endless metrics that could be taken from even a simple simulation sequence such as this.

2. Write a Swing Appointment Reminder Application [30 points]

Our Appointment Reminder Application uses an external calendar widget which we found on GitHub. The whole application uses the Gradle build system to manage external dependencies and building the application.

The program is centered around the AppointmentCalendar.java file which builds all the component panels. Using the observer-observable pattern, panels can communicate between one another to update components of the application. The three main panels are the panel containing the calendar, the panel with the appointment information, and the panel with buttons which modify the frame.

The calendar widget in the CalendarPanel updates the ListView of the AppointmentPanel. The AppointmentPanel contains all the appointments in the AppointmentBook for a given day and can be modified by clicking on a different date in the calendar. From the available options for a given day, you can select an appointment from the ListView, and the information will be displayed in a form beside the ListView. The button panel allows a user to hide and show the appointment panel, and close the application by invoking the Top-level windows’ exit procedure.

Our application allows a user to add or remove an appointment using a button which gives the user a popup with fields to fill, or a remove button which deletes the current selection.

Some of the big hurdles to overcome during the development of this application dealt with establishing the hierarchy between the panels. This was important, since the observers needed to be able to change characteristics of panels that they were not members of. Since the entire application runs on an event queue it was also challenging to ensure that each component maintained the AppointmentBook invariant, and only accessed/modified valid appointments. Failure to do so would cause other components to stop working properly as the threads hit unhandled errors.

We found the best solution to overcome this problem was to use the ListView as the driver for making modifications to the application, and using all the other components as either inputs or outputs for what is in the View. As a result, we prevented simultaneous operations from occurring which seemed to have unexpected side effects.

3. UML Diagram

Our UML diagram explains the relationships in a Library Management System. The main components are Employees, Books, Accounts, and Borrowers. The employees are Librarians of which we have different types. There is a Head Librarian, regular librarians, volunteers, and students. A librarian is a person and has personal information such as a name and an address.

Our library has books which are composed of a media type, checkout time/due date, whether the book is overdue or not, a title, ISBN, and a condition for the book. There are different media types such as books, magazines, periodicals, newspapers, etc. The condition captures whether the book is in good condition, fair condition, has water damage, or some other damage that makes the book unusable.

Our accounts are composed of an ID, date the account was opened, status of the account meaning whether the account is active, inactive, or flagged (if the account has an overdue book), and the history of the books that have been checked out to this account.

A borrower is a person who has a name and an address. They also have a borrower status which reflects their membership level. The membership level controls how many books a borrower can checkout, the checkout duration, and which items in the library can be checked out (books only, database, or online). A borrower has an account which lets them checkout books from the library of which the library is composed.

The major obstacles that we encountered while designing our UML class diagram were in understanding the class relationships between the separate components. This was overcome when we started to fill in the types for the member attributes for each component. Also, the program that we chose to layout our diagram required a good deal of finesse and patience. This was largely overcome due to team members having previous experience using the UML tool.