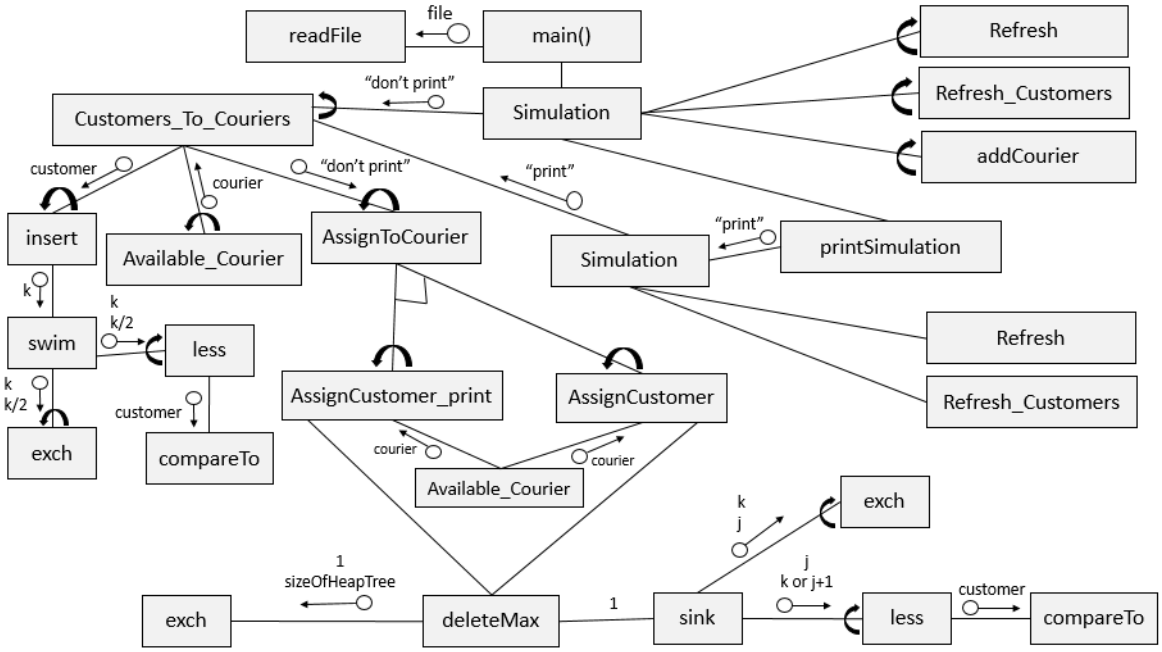
Basme Zantout - Zeynep Sude Bal **Report**

Question 1

➔ Problem Statement and Code Design



➔ Implementation, Functionality

We have 4 classes for this project as Customer.java, Courier.java, Scheduling.java and Main.java.

- **Courier.java**

  We basically implemented the ***Courier.java class***. It has 2 attributes as ***ID*** which is the ID of the Courier and ***finish_time*** which is the time to pick up a new order when the courier is available.

- **Customer.java**

  It has 5 attributes as ***ID*, *Year*, *order_time*, *service_time*, and *wait_time***. For every Customer object, we define its ID, year, order time and service time. Also, the attribute ***wait_time*** is the time elapsed between the customer's specified order time and the time he is assigned to a courier.

  1. **compareTo Method:**

     As we implement ***Comparable*** interface in Customer.java class, we need to implement ***compareTo*** method so that we can identify the Customer's priorities. First of all, it determines the priorities of customers by comparing their registration year. Priority is given to the client who has the oldest registration. In the second place, if the customer's registration years are identical, then it compares their waiting time, and who has the greater waiting

time has the priority. Afterward, if both the year and the waiting times are same, then the customer with the lower ID number takes precedence.

- **Scheduling.java**

  This class has two more classes inside it as ***Node*** which is implemented to be used by the ***ListOfCouriers*** class. In ListOfCouriers, we add a new Courier node to the linked list of Couriers by using ***addCourier*** method.

  1. **sink():** It basically carries out sink operation when removing an element from the Heap-based array. It calls the ***less()*** method so that it can compare the precedence of two Customer objects which are the children of the root element. When the comparing process is done, it calls the exch***()*** method to swap the root element and its child having less priority if the root has less priority than the child.
  2. **deleteMax():** In this method, we first swap root element and last element in our heap tree so that we can remove root from the tree as it has the maximum value. We call the ***exch()*** method for swapping operations and then, we call ***sink()*** method in order to perform a sink operation in the tree.
  3. **swim():** It basically carries out swim operation when inserting an element in the Heap-based array. It calls the less***()*** method so that it can compare the added element to its parent. Swapping operation is carried out by calling ***exch()*** if the added Customer has a higher priority than its parent.
  4. **exch():** This method takes two Heap-based array index values as a parameter and swaps the entries at those indices.
  5. **less():** It compares two Customer objects with respect to their priority by calling ***compareTo()*** method in **"*Customer.java*"** class.
  6. **insert():** It basically adds a new Customer to the end of the ***"Customers HeapTree"*** which is a Heap-Tree-based priority queue array and then calls ***swim()*** method to conduct a swim process.
  7. **AssignCustomer():** This method is for assigning customers to couriers. We call ***deleteMax()*** method in order to delete customers from the Heap Tree. Then, we calculate the waiting time of the deleted customer by subtracting the finish time of the available courier and the order time of the deleted customer. We call the ***Available_Courier()*** method in order to get a courier who is available at that moment. Also, we add the waiting time we calculated to sum all waiting times.
  8. **Available_Courier():** It checks the availability of couriers. If more than one courier completes the order at the same time, the one with the lowest ID receives the order.
  9. **AssignCustomer_print():** The operation is similar to ***AssignCustomer()***, but this one also prints the output to the console.
  10. **AssignToCourier():** It assigns customers to couriers at a time. After the assigning process is done, we allocate as many customers as there are left for remaining customers in the Heap-Tree array. It specifies whether or not the

simulation should be printed according to its parameter. If yes, it calls ***AssignCustomer_print()***, otherwise, it calls ***AssignCustomer()***.

11. **Customers_To_Couriers():** It iterates over each customer in the ***"customers waiting"*** which is an array holding all Customers. Afterwards, it adds customers who place orders during available times of couriers to the Heap-based array and assigns them to available couriers. We also call ***AssignToCourier()*** by sending a String parameter to identify whether the simulation has to be printed or not.
12. **Simulation():** By using the most recent and optimal number of Couriers computed, it carries out the simulation. It calls ***Refresh_Customers()*** to reset the customers' information and calls ***Refresh()*** from ***ListOfCouriers.java*** class to refresh the couriers' data. It determines that the simulation should be printed or not by calling the ***Customers_To_Couriers()*** method and passing its parameter to it.
13. **printSimulation():** It outputs the results of the minimum number of couriers needed, the simulation with that number of couriers, and the average waiting time which is computed.
14. **Simulation():** The method begins the simulation within a loop by one courier and required customers are assigned to the available courier in terms of their priority by calling ***Customers_To_Couriers()*** method. Also, we calculate the average waiting time of all the customers inside the loop. The loop is iterating until we achieve an average waiting time less than or equal to the maximum average waiting time. When we get the required average waiting time, the iteration of the loop ends. Then we call ***printSimulation()*** method to print the simulation output to the console.

- **Main.java**
    1. **main():** After we read the file by calling the ReadFile***()*** method, we call the **Simulation*()*** method from the ***Scheduling.java*** class.
    2. **ReadFile():** It takes the file name as a parameter and reads a text file of integer values.

➔ **Testing**

We tested the program by different maximum average waiting time values and we ran it by using various numbers of customers. Also, we control the program in order to determine whether it outputs correctly if we change the customer's ID and waiting times.

➔ **Final Assessments**

The challenging part in this assignment was figuring out the question properly. At first, we were confused and had trouble understanding the question and how to approach it, and how to implement it while we are coding. Therefore, we divided the question into sections and started coding step by step and it became easier in this way. On the other hand, what we liked about the assignment is that we had a chance to consolidate Heap and Priority Queue subjects in terms of their implementation.