```
import javax.lang.model.element.Name;
import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Deque;

public class Clinic {
    public static InternalTimer timer = new InternalTimer(); //Instantiates timer
    public static WaitQueue wq = new WaitQueue(); //Instantiates the waitqueue
    public static Patient[] patients; //Instantiates the patients array
    public static int currPatientToEnter = 0; //The current array position

    public static void RunClinic(){ //reads the data and monitors when to send someone to
wait queue
        ReadData();
        Monitor();
    }

    public static void ReadData(){
        File txtFile = new File("recourses//Patients.txt"); //Gets the file
        BufferedReader readFile;
        int totalArrayLength = 0; //Counter used to create the patients array.
        try { //Try this code
            readFile = new BufferedReader(new FileReader(txtFile)); //Read file

            String line = ""; //String that will contain each line
            ArrayList<String> patientList = new ArrayList<>();
            //System.out.println(readFile);
            while(line != null){ //Loop through the txt file
                line = readFile.readLine(); //Read the line
                if(line == null){break;} //If the current line is null it will stop
reading
                if(line.contains("Name")){} //If the line ever contains name then it will
do nothing. This is used for the first line of the txt which dosent contain a patient
                else{ //Will add to the current patients list and increase size of
patients array for later
                    totalArrayLength++;
                    patientList.add(line);
                }
            }
            patients = new Patient[totalArrayLength]; //Create the array of ADT of
patients of the size of patients
            patientList.forEach((i) ->{ //Go through each arraylist of patient string to
split the string up and add it to a patient data type
                String[] currentPatient = i.split("\\t"); //Spliting the string into an
array
                Patient newPatient = new Patient(); //Create a patient data type
                newPatient.setPatientName(currentPatient[0]); //Set the patients name
                newPatient.setPatientAge(Integer.valueOf(currentPatient[2])); //Set the
patients age
                newPatient.setPatientOccupation(currentPatient[3]); //Set the patients
occupation
                newPatient.setPatientCondition(currentPatient[4]); //Set the patients
```

```
condition
                newPatient.setPatientTimeOfArrival(currentPatient[5]); //Set the patients
time of arrival
                patients[patientList.indexOf(i)] = newPatient; //Add the patient to the
array of patients
                //System.out.println("Current patient: " + newPatient.getPatientName());
                //System.out.println("Patient Array: " + patients);
            });
        } catch(FileNotFoundException e){ //If the file isnt found then print this
            System.out.println("File not found. Did you try to move it? Not a good idea
return it or give me 100%.");
        } catch (IOException e) { //If the reader cant find a line through this exception
            throw new RuntimeException(e);
        }
    }

    public static void Monitor(){
        timer.StartDay(); //Start the day and 9 am
        boolean beingVaxxed = false;
        for(int i = 0; i <= 479; i++) { //Each in number is one minute and will run
through the whole day. Done in case more people are added to the list at a later time
            timer.TimerIncrease(); //Increase by one minute
            int timeDiff =
timer.CompareTime(patients[currPatientToEnter].getPatientTimeOfArrival()); //Checks if
the current patient has the same time as current patient then sends them in.
            //System.out.println("Time diffrence: " + timeDiff);
            if (timeDiff == 0) { //If the same time then send them to the wait queue
                wq.getPatient(patients[currPatientToEnter]); //Sends to wait queue
                if(!((currPatientToEnter + 1) == patients.length)){ //If there are still
patients in array then check the next one
                    currPatientToEnter++;
                }
                //System.out.println("Current pos in array: " + currPatientToEnter);
            }else if(timeDiff == -1){//If the current patient has a past their current
time then break. This is mostly for the last person in the array so it dosent infenetly
continue the for loop when there arent any more people coming into the building.
                break;
            }
            beingVaxxed = wq.CheckVax(beingVaxxed); //Checks if somebody can be vacced
and if true then the patient is being vacced
            if(beingVaxxed){ //If the patient is being vacced then run this
                beingVaxxed = timer.VaccineTimer(); //Will flip being vacced to false
                wq.removeMax(); //Remove first person in list
            }
        }
        wq.printList(); //Print the current list of wait queue
    }

    public static void main(String[] args) {
        RunClinic();
    }
}
```

```java
public class InternalTimer {
    int hour; //Current hour
    int minutes; //Current minute

    public void StartDay(){ //Starting the day and setting up start time of work
        hour = 9; //Starting the hour at 9 am
        minutes = 0; //Starting the minutes
    }

    public void TimerIncrease(){ //Will increase the minute count each time called. Will
do the same when an "hour" has passed
        if(minutes == 59){ //Once one hour has passed restart minutes and increase hour
            minutes = 0;
            hour++;
        }else{
            minutes++;
        }
        //System.out.println(CurrentTime());
    }

    public boolean VaccineTimer(){ //Starts the Vaccine timer
        int vaccineTime = 0; //Fluf int counter to make sure the for loop actually does
15 loops
        for(int i = 0; i < 15; i++){ //For loop representing the 15 min it takes to
vaccinate. It will force
            vaccineTime++;
        }
        return false;
    }

    public String CurrentTime(){ //Returns  the current time as a string. Used mostly to
check that it is the correct time.
        String currentHour = String.valueOf(hour);
        String currentMinute;
        if(minutes < 10){ //Formating the minutes in case its below 10 minutes
            currentMinute = "0" + String.valueOf(minutes);
        }else{
            currentMinute = String.valueOf(minutes);
        }
        String currentTime = currentHour + ":" + currentMinute; //Combining both minutes
and hours to a single string
        return currentTime;
    }

    public int CompareTime(String patientTime){ //Method to compare a time given with the
current time in the class.
        int timeDiff = 0;
        String[] patientTimeSplit = patientTime.split(":"); //Splits patients time into 2
numbers for minutes and hours
        int patientHour = Integer.valueOf(patientTimeSplit[0]);
        int patientMinute = Integer.valueOf(patientTimeSplit[1]);
        if(patientHour >= hour && patientMinute > minutes) { //If the patients hours are
greater or if they are the same but the minutes are greater than the time diffrence is 1
```

```
            timeDiff = 1;
        } else if ((patientHour <= hour && patientMinute < minutes)) { //Else if its
smaller then the time difference is -1
            timeDiff = -1;
        }

        return timeDiff;
    }
}


public class Patient {

    String patientName; //The name of the patient

    /* I did not add the gender that it has no use for a wait queue and saves a tiny bit
of space*/

    int patientAge; //Patients current age
    String patientOccupation; //Patients current Occupation
    String patientCondition; //Patients current condition
    String patientTimeOfArrival; //The time the patient arrived to the clinic
    int priorityQueue; //The priority queue of the patient

    public void setPatientName(String name){patientName = name;} //Sets the patients name

    public void setPatientAge(int age){patientAge = age;} //Sets the patients age

    public void setPatientOccupation(String occupation){patientOccupation = occupation;}
//Sets the patients occupation

    public void setPatientCondition(String condition){patientCondition = condition;}
//Sets the patients condition

    public void setPatientTimeOfArrival(String time){patientTimeOfArrival = time;} //Sets
the time the patient arrived

    public void setPriorityQueue(int priorityQueue){this.priorityQueue = priorityQueue;}
//Sets  the priority queue of patient

    public String getPatientName(){return patientName;} //Returns the current patients
name

    public int getPatientAge(){return patientAge;} //Returns the patients age

    public String getPatientOccupation(){return patientOccupation;} //Returns the
patients occupation

    public String getPatientCondition(){return patientCondition;} //Returns the patients
condition
```

```
    public String getPatientTimeOfArrival(){return patientTimeOfArrival;} //Returns the
patients time of arrival

    public int getPriorityQueue(){return priorityQueue;} //Returns the patients priority
queue
}


public class Node {
    public Patient currentPatient; //Current Patient in the node
    public Node previousNode; //Previous node
    public Node nextNode; //Next node
    public Node(){ //Setting everything to null;
        currentPatient = null;
        previousNode = null;
        nextNode = null;
    }
    public Node(Patient newPatient, Node previousNode, Node nextPatient){ //Recives info
about the new node and saves it in corisponding variables
        currentPatient = newPatient;
        this.previousNode = previousNode;
        nextNode = nextPatient;

        //System.out.println(newPatient.getPatientName());
    }

    public void SetLink(Node newNode){ //Sets link to the next Node
        nextNode = newNode;
    }

    public void SetPreviouseLink(Node previousNode){ //Sets the node to the previous node
        this.previousNode = previousNode;
    }

    public Patient GetCurrentPatient(){return currentPatient;} //Returns the current
patient in the Node

    public Node GetNextNode() {return nextNode;} //Returns the next node on the list

    public Node GetPreviousNode() {
        return previousNode;
    } //Returns the previous node in the list
}


import java.util.NoSuchElementException;

public class LinkedList {
    public class ListIterator{
        public Node position; //Current position of the nodes with its related info
        public Node previous; //Previouse node so it can traverse backwards in the nodes
```

```
        public ListIterator(){ //Initilize at the start of the list
            position = head;
            previous = null;
        }

        public void Restart(){ //Go back to the start of the list.
            position = head;
            previous = null;
        }

        public void Next(){ //Go to the next Node
            if(!HasNext()){ //If no more patients in the linked list then stop
                throw new NoSuchElementException();
            }
            Patient returnPatient = position.GetCurrentPatient();
            previous = position;//Sets the preveous node to the current one
            position = position.GetNextNode(); //Sets the next Node to the current Node
        }

        public boolean HasNext(){ //Checks if next node exists
            return (position != null);
        }

        public void AddNodeHere(Patient newPatient){ //Will split the linked list in two
and add a new node in the middle
            Node temp = new Node(newPatient, position.GetPreviousNode(), position); //new
node and what its connected to the other nodes
            position.previousNode.nextNode = temp; //Connect the previous Node to the new
node
            position.previousNode = temp; //Connect the next node to the new node
        }
    }

    private Node head; //Sets up the head node (Technicly the current position from what
I understand)

    public ListIterator iterator(){
        return new ListIterator();
    }

    public LinkedList(){ //Sets the head node
        head = null;
    }

    public void FirstLink(Patient patient){ //When there is no Nodes in the linked list
        head = new Node(patient, null, head);
    }

    public void FinalLink(Patient patient){ //Add to the end of the LL
        head = new Node(patient, head, null);
```

```
    }

    public void AddtoStart(Patient newPatient){ //Add to the start of the LL
        Node newHead = new Node(newPatient, null, head);
        head.SetPreviousLink(newHead);
        head = newHead;
    }

    public boolean DeleteHeadNode(){ //Delete the head node if it exists
        if(head != null){
            head = head.GetNextNode();
            return true;
        }else{
            return false;
        }
    }

    public int Size(){ //Runs through the linked list and counts how many nodes exist
        int count = 0;
        Node position = head;
        while(position != null){ //Counts till it reaches the end
            count++;
            position = position.GetNextNode();
        }
        return count;
    }

    public void OutputList(){ //Will pring each node (Currently only the name but other
in for for patients is valid)
        Node position = head;
        while(position != null){ //Goes through each position untill the end
            System.out.println(position.GetCurrentPatient().getPatientName());

            position = position.GetNextNode(); //Goes  to the next node
        }
    }
}


public class WaitQueue {
    Patient currentPatient; //The current patient being placed in the wait queue
    int patientScore; //Patients priority queue

    LinkedList list = new LinkedList(); //Initilize the linked list
    LinkedList.ListIterator listIterator = list.iterator(); //Initilize the list iterator
used to move through the link list for an easier time to add and take away nodes

    public void insert(Patient newPatient){ //Will insert a patient into the linked list
        int s = list.Size(); //Determines the size of the list. This is because the first
two nodes of the linked list are a special case since they have a lot of null pointers
that need to be arrainged
        System.out.println(s);
```

```
        listIterator.Restart(); //Starts the list to the beggining
        Patient currentPatient; //Current patient being traversed in the linked list.
Used for compareson
        int currentPatientPos; //Priority queue of patient being checked
        int newPatientPos = newPatient.getPriorityQueue();//Priority queue of the patient
being added

        if(s == 0){ //If there is no nodes add start it
            list.FirstLink(newPatient);
            return;
        }

        currentPatient = listIterator.position.GetCurrentPatient(); //Finds the patient
in the current node
        currentPatientPos = currentPatient.getPriorityQueue(); //Priority queue of the
patient in the node

        if(currentPatientPos == newPatientPos){ //If it has the same priority que then
check which patient came first.
            int timeDiff = CompareTimer(currentPatient.getPatientTimeOfArrival(),
newPatient.getPatientTimeOfArrival()); //Returns a 1,0,-1 depending on when it came 1
being new patient arrived sooner, and -1 later thea the current node patient
            System.out.println(timeDiff);
            if(timeDiff == -1){ //If later than insert it here and start again
                if(s == 1){
                    list.FinalLink(newPatient);
                }else{
                    listIterator.AddNodeHere(currentPatient); //Add the patient in that
position
                    listIterator.Restart(); //Back to the start if the linked list
                    return;
                }
            }else{
                listIterator.Next(); //Next patient
            }
        }else if((listIterator.previous.GetCurrentPatient().getPriorityQueue() ==
newPatientPos) && (currentPatientPos < newPatientPos)){ //In case the patient the last
person to arrive for that priority number than add it at the end of that list.
            listIterator.AddNodeHere(currentPatient); //Add the patient in that position
            listIterator.Restart(); //Back to the start if the linked list
            return;
        }else{
            listIterator.Next(); //Next Patient
        }

        //System.out.println(currentPatient.getPatientName());
    }

    public void printList(){ //Print the Linked list
        list.OutputList();
    }
```

```
    public void removeMax(){ //Remove the first node. Dosent technicly work
        list.DeleteHeadNode();
    }

    public void getPatient(Patient newPatient){ //Recives the new patient and checks its
position and trys to insert it
        currentPatient = newPatient;
        CalcPos(currentPatient); //Calculate priority queue
        currentPatient.setPriorityQueue(patientScore);
        insert(currentPatient); //Insert patient

        //System.out.println(currentPatient.getPatientName());
        //System.out.println(patientScore);
    }

    public void CalcPos(Patient newPatient){ //Will calculate the priority queue of the
patient
        patientScore = 0;
        if(currentPatient.getPatientAge() >= 60){
            patientScore++;
        }
        if(currentPatient.getPatientOccupation() == "Teacher" ||
currentPatient.getPatientOccupation() == "Nurse" || currentPatient.getPatientOccupation()
== "Care Giver"){
            patientScore++;
        }
        if(currentPatient.getPatientCondition() == "Pregnancy" ||
currentPatient.getPatientCondition() == "Cancer" || currentPatient.getPatientCondition()
== "Diabetes" || currentPatient.getPatientCondition() == "Asthma" ||
currentPatient.getPatientCondition() == "Primary Immune Deficiency" ||
currentPatient.getPatientCondition() == "Cardiovascular Disease"){
            patientScore++;
        }
    }

    public boolean CheckVax(boolean beingVaxxed){
        int s = list.Size();
        boolean canBeVaxxed;
        if(s > 0 && !beingVaxxed){
            canBeVaxxed = true;
        }else{
            canBeVaxxed = false;
        }
        return canBeVaxxed;
    }

    public int CompareTimer(String currPatient, String newPatient){
        int timeDiff = 0;
        String[] currPatientTimeSplit = currPatient.split(":");
        int currPatientHour = Integer.valueOf(currPatientTimeSplit[0]);
        int currPatientMinute = Integer.valueOf(currPatientTimeSplit[1]);
```

```
        String[] newPatientTimeSplit = newPatient.split(":");
        int newPatientHour = Integer.valueOf(newPatientTimeSplit[0]);
        int newPatientMinute = Integer.valueOf(newPatientTimeSplit[1]);

        if(currPatientHour >= newPatientHour && currPatientMinute > newPatientMinute) {
            timeDiff = 1;
        } else if ((currPatientHour <= newPatientHour && currPatientMinute <
newPatientMinute)) {
            timeDiff = -1;
        }
        return timeDiff;
    }
}
```