```
1 public class Main {
2     public static void main(String[]
  args) throws Exception{
3         Clinic c1 = new Clinic();
4         c1.Monitor();
5     }
6 }
7
```

```
 1 class Node {
 2     Patient PatientData; // contains
   patient instance as data
 3     int priorityLevel; // contains
   priority level
 4     Node prev; // previous node
 5     Node next; // next node
 6 
 7     public Node(Patient PatientData,
   int priorityLevel){
 8         this.PatientData =
   PatientData;
 9         this.priorityLevel =
   priorityLevel;
10         this.prev = null;
11         this.next = null;
12     }
13     public Node(Patient PatientData,
   int priorityLevel, Node prev, Node
   next){
14         this.PatientData =
   PatientData;
15         this.priorityLevel =
   priorityLevel;
16         this.prev = prev;
17         this.next = next;
18     }
19 }
20 
```

```
 1 import java.util.Comparator;
 2
 3 public class Timer{
 4     Clinic testClinic = new Clinic();
 5     int hour;
 6     int minute; // resets every 60
   minutes
 7     int vxtime; // resets every 15
   minutes // required time to be
   vaccinated.
 8     String fStr;
 9     String str, str1;
10
11     String []temp;
12
13     public Timer(){
14         temp = testClinic.patients[1
   ].getTimeOfArrival().split(":");
15
16         this.hour = Integer.parseInt(
   temp[0]);
17         this.minute = Integer.
   parseInt(temp[1]);
18         this.vxtime = 0;
19     }
20
21     public boolean increase(){
22
23         this.minute++;
```

```
24          this.vxtime++;
25
26          if(this.minute == 60) {
27              this.hour++;
28              this.minute = 0;
29          }
30
31          if(this.vxtime >= 15) {
32              this.vxtime = 0;
33              return true; // remove
    from clinic
34          }
35          else
36              return false; // is In
    clinic
37      }
38
39      public int compare(String
    clinicData) {
40
41          /**  if hour is less than
    adds 0 before the digit else keeps
    the same value  */
42
43          this.fStr = convertToString(
    this.hour, this.minute);
44
45          if(clinicData.equals(fStr))
46              return 0;
```

```
47          else if(fStr.compareTo(
   clinicData) < 0)
48              return -1;
49          else
50              return 1;
51      }
52
53      public int getHour() {
54          return hour;
55      }
56
57      public int getMinute() {
58          return minute;
59      }
60
61      public String convertToString(int
    hour, int minute){
62          if(getHour() < 10)
63              str = "0" + this.hour;
64          else
65              str = "" + this.hour;
66
67          /**  if minute is less than
   adds 0 before the digit else keeps
   the same value  */
68          if(getMinute() < 10)
69              str1 = "0" + this.minute;
70          else
71              str1 = "" + this.minute;
```

```
72          return str + ":" + str1;
73      }
74
75      public String toString(){
76          return convertToString(this.
    hour, this.minute);
77      }
78  }
79
```

```
 1 //Clinic does the
 2 import java.io.File;
 3 import java.io.FileNotFoundException;
 4 import java.util.Scanner;
 5
 6 public class Clinic{
 7
 8     public Patient [] patients = new
   Patient[17];
 9     WaitQueue wq = new WaitQueue();
10
11     public Clinic(){
12         try{
13             readData();
14         }catch(FileNotFoundException
   e){
15             System.out.println("Error
   : File Not Found.");
16         }
17     }
18
19     public void readData() throws
   FileNotFoundException {
20         //reads data from file
21         File readFile = new File("H:
   \\DataStructure\\COSC2P03_A1_v2\\
   COSC2P03_A1_v2\\patients.txt");
22         Scanner input = new Scanner(
   readFile);
```

```
23
24          int i = 0;
25          while(input.hasNextLine()){
26              String s = input.nextLine
    ();
27              patients[i] = new Patient
    (s);
28              i++;
29          }
30      }
31
32      public void Monitor(){
33          //WaitQueue Class
    functionality used in here.
34          //Also uses Timer class
    Functionality.
35
36          /**
37           * patients added and removed
     in this function.
38           * timer class is used for
    incrementing time and for checking
    whether
39           * a patient is ready to
    enter the queue or not.
40           */
41
42          Timer timer = new Timer();
43
```

```
44          /**
45           * Calculates Time for clinic to run based first patient's arrival
46           * and last patient's arrival;
47           * + 15 added -> is for vx time for the last patient.
48           */
49
50          int TimeForClinicRun = (patients.length-1) * 15;
51
52
53          int k = 1;
54          for(int i = 0; i < TimeForClinicRun; i++){ //problematic logic
55              if(k <= 15 && timer.compare(patients[k].getTimeOfArrival()) == 0
56                      ){ //Inserts patients in the queue.
57                  wq.insert(patients[k], calculatePriorityLevel(patients[k]));
58                  k++;
59              }
60              if(timer.increase() && wq != null){ //Removes Patients from
```

```
60 queue.
61                     try{
62                         System.out.
   println("Patient Name: " + wq.
   getPeekItem().PatientData.
   getPatientName()
63                                 + "
   Arrival Time: " + wq.getPeekItem().
   PatientData.getTimeOfArrival()
64                                 + " Check
    Out Time: " + timer);
65                         wq.removeMax();
66                     }catch(
   NullPointerException e){
67                         System.out.
   println(" ");
68                     }
69                 }
70                 if( k >= 16)
71                     patients = null;
72             }
73         }
74
75         public int calculatePriorityLevel
   (Patient patient){
76             int pL = 0;
77
78             if(patient.getAge() >= 60)
79                 pL++;
```

```
80          if(patient.getOccupation().
    equals("Teacher") ||
81              patient.getOccupation().
    equals("Nurse") ||
82              patient.getOccupation().
    equals("Care Giver"))
83                pL++;
84          if(patient.
    getHealthCondition().equals("
    Pregnant") ||
85              patient.
    getHealthCondition().equals("Cancer"
    ) ||
86              patient.
    getHealthCondition().equals("
    Diabetes") ||
87              patient.
    getHealthCondition().equals("Asthma"
    ) ||
88              patient.
    getHealthCondition().equals("Primary
     Immune Deficiency") ||
89              patient.
    getHealthCondition().equals("
    Cardiovascular Disease"))
90                pL++;
91
92          return pL;
93      }
```

```
94 }
```

```
 1 public class Patient {
 2 
 3     private String patientName;
 4     private String gender;
 5     private int age;
 6     private String occupation;
 7     private String healthCondition;
 8     private String timeOfArrival;
 9 
10     public Patient(){
11         patientName = null;
12         gender = null;
13         age = 0;
14         occupation = null;
15         healthCondition = null;
16         timeOfArrival = null;
17     }
18 
19     public Patient(String dataStream
   ) throws NumberFormatException{
20         String []temp = new String[6
   ];
21         temp = dataStream.split("\t"
   );
22 
23         setPatientName( temp[0] );
24         setGender( temp[1] );
25         try{
26             setAge( Integer.parseInt(
```

```
26 temp[2]) );
27          }catch (NumberFormatException
    e ){
28              setAge(0);
29          }
30          setOccupation( temp[3] );
31          setHealthCondition( temp[4
   ] );
32          setTimeOfArrival( temp[5] );
33      }
34
35      public String getPatientName() {
36          return patientName;
37      }
38
39      public void setPatientName(String
    patientName) {
40          this.patientName =
   patientName;
41      }
42
43      public String getGender() {
44          return gender;
45      }
46
47      public void setGender(String
   gender) {
48          this.gender = gender;
49      }
```

```
50
51      public int getAge() {
52          return age;
53      }
54
55      public void setAge(int age) {
56          this.age = age;
57      }
58
59      public String getOccupation() {
60          return occupation;
61      }
62
63      public void setOccupation(String
    occupation) {
64          this.occupation = occupation;
65      }
66
67      public String getHealthCondition
    () {
68          return healthCondition;
69      }
70
71      public void setHealthCondition(
    String healthCondition) {
72          this.healthCondition =
    healthCondition;
73      }
74
```

```
75      public String getTimeOfArrival
    () {
76          return timeOfArrival;
77      }
78
79      public void setTimeOfArrival(
    String timeOfArrival) {
80          this.timeOfArrival =
    timeOfArrival;
81      }
82
83      public String toString(){
84          return "Patient Name: [ " +
    patientName + " Gender: " + gender
     + " Age: " + age +
85                  " Occupation: " +
    occupation + " Health Condition: "
     + healthCondition +
86                  " TimeOfArrival: "
     + timeOfArrival + " ]";
87      }
88 }
```

```
 1 //Usage of Priority Queue Here.
 2 public class WaitQueue {
 3     //Attributes
 4     Node _dll_head;
 5     Node front, rear;
 6     int priorityLevel;
 7 
 8     public WaitQueue(){
 9         _dll_head = null;
10         front = rear = null;
11         priorityLevel = 0;
12     }
13 
14     public Node removeMax() {
15         Node temp;
16         //remove Max using DL_List
   based on queue
17         if(front.next == null){
18             front = null;
19             _dll_head = null;
20         }
21         else{
22             front = front.next;
23             front.prev = null;
24             _dll_head = front;
25         }
26         temp = front;
27 
28         return front;
```

```
29      }
30
31      /**
32       * @param dataStream -> receives
   data from patients array.
33       * @param priorityLevel ->
   assigns priority based on data
34       */
35      public void insert(Patient
   dataStream, int priorityLevel){
36          //insert using DL_List based
   on queue
37          Node newNode = new Node(
   dataStream, priorityLevel);
38          Node p;
39
40          if(_dll_head == null || front
    == null) {
41              _dll_head = newNode;
42              front = rear = _dll_head;
43          }
44          else if(priorityLevel > front
   .priorityLevel){
45              newNode.next = front.next
   ;
46              if(front.next != null){
47                  front.next.prev =
   newNode;
48              }
```

```
49              front.next = newNode;
50              newNode.prev = front;
51          }
52          else{
53              p = front;
54
55              while(p.next != null){
56                  if(priorityLevel > p.
   priorityLevel){
57                      break;
58                  }
59                  p = p.next;
60              }
61              if(priorityLevel > p.
   priorityLevel){
62                  newNode.prev = p.prev
   ;
63                  if(p.prev != null){
64                      p.prev.next =
   newNode;
65                  }
66                  p.prev = newNode;
67                  newNode.next = p;
68              }
69              else if(priorityLevel <=
   rear.priorityLevel &&
70                      ( dataStream.
   getTimeOfArrival().compareTo(rear.
   PatientData.getTimeOfArrival()) > 0
```

```
70  )){
71                  rear.next = newNode;
72                  newNode.prev = rear;
73                  rear = newNode;
74              }
75              else{
76                  p = front;
77
78                  while(p.next != null
    && priorityLevel == p.priorityLevel
    &&
79                          dataStream.
   getTimeOfArrival().compareTo(p.
   PatientData.getTimeOfArrival()) < 0){
80                      p = p.next;
81                  }
82
83                  if(dataStream.
   getTimeOfArrival().compareTo(p.
   PatientData.getTimeOfArrival()) < 0){
84                      //adding node
   before
85                      newNode.next = p.
   next;
86                      if(p.next != null
   )
87                          p.next.prev
    = newNode;
88                      p.next = newNode;
```

```
 89                         newNode.prev = p
    ;
 90                     }
 91                     else{
 92                         //adding node
    after.
 93                         newNode.next = p
    .next;
 94                         if(p.next !=
    null){
 95                             p.next.prev
     = newNode;
 96                         }
 97                         p.next = newNode
    ;
 98                         newNode.prev = p
    ;
 99                     }
100
101                 }
102             }
103     }
104
105     public Node getPeekItem(){
106         return _dll_head;
107     }
108 }
```