**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**



**Subject Name: Object Oriented Programming**

**Complex Engineering Problem Report**

**Submitted To:**

**Professor Anum Abdul Salam/ LE Emaan Fatima**

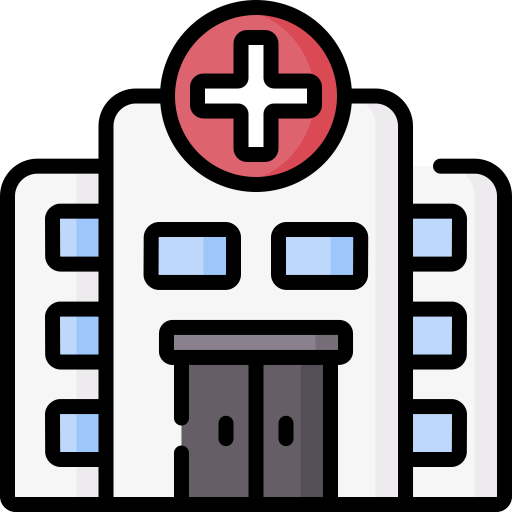
**Submitted By:**

**Usman Awan REG # 480836**

**DE - 45/SYN:A DEPT: Computer Engineering**

**Submission Date: 01/01/2025**

**Title: Hospital Emergency Department System**



**Table of contents:**

|  |  |  |
| --- | --- | --- |
| **S. no.** | **Contents** | **Page** |
| 1. | Emergency Department(ED) ward overview | 3 |
| 2. | Introduction to Problem | 4 |
| 3. | Objectives | 5 |
| 4. | Libraries and tools used | 5 |
| 5. | Design overview - UML | 6 |
| 6. | Source code overview   * Patient class * ResourcePool class * SimulationManager class * SimulationGUI class | 7 |
| 7. | Key features | 19 |
| 8. | Primary challenges | 19 |
| 9. | Functional overview - Packaging and Simulations | 20 |
| 10. | Conclusion and credits | 21 |

**Emergency Department(ED) ward overview:**

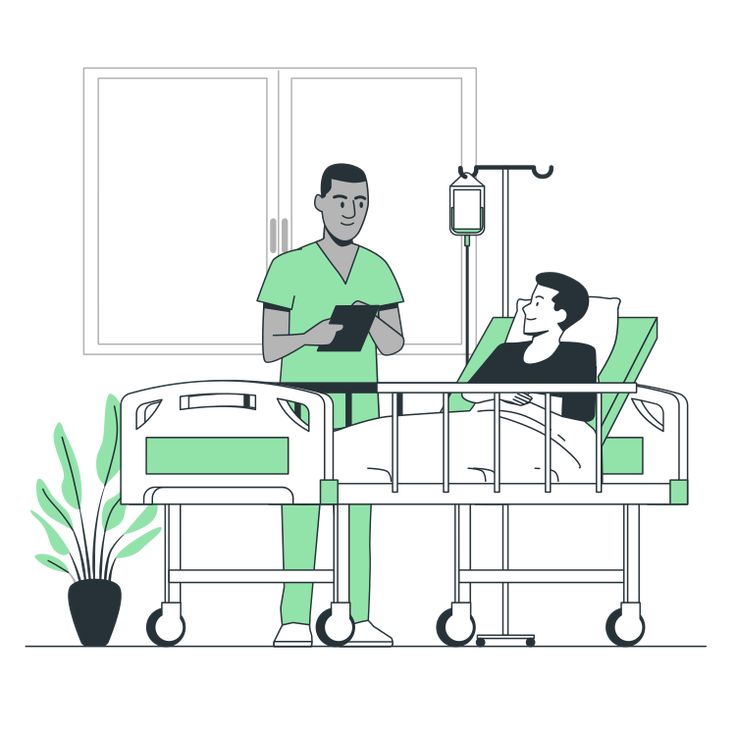
An **emergency department** (**ED**), also known as an **accident and emergency department** (**A&E**), **emergency room** (**ER**), **emergency ward** (**EW**) or **casualty department**, is a medical treatment facility specializing in [emergency medicine](https://en.wikipedia.org/wiki/Emergency_medicine), the [acute](https://en.wikipedia.org/wiki/Acute_(medicine)) care of patients who present without prior appointment; either by their own means or by that of an [ambulance](https://en.wikipedia.org/wiki/Ambulance). The emergency department is usually found in a [hospital](https://en.wikipedia.org/wiki/Hospital) or other [primary care](https://en.wikipedia.org/wiki/Primary_care) center.

Due to the unplanned nature of patient attendance, the department must provide initial treatment for a broad spectrum of illnesses and injuries, some of which may be [life-threatening](https://en.wikipedia.org/wiki/Medical_emergency) and require immediate attention. In some countries, emergency departments have become important entry points for those without other means of access to medical care.

The emergency departments of most hospitals operate 24 hours a day, although staffing levels may be varied in an attempt to reflect patient volume.

**----------------------------------------------------------------------------------------------------**

**Introduction to Problem:**

Emergency Department(ED) of a hospital is a specialized ward to deal semi-critical and severely critical cases. The ED ward consists of a finite number of beds and a waiting dorm for patients when all beds are occupied. Surgeons and internee surgeons treat the patients with specific ratios for doctors to internee doctors for each patient according to the severity. Each patient goes through the triage stage to get severity analysis and further operations.

Weibull distribution is used to calculate the time interval between patient arrivals and the time taken for patient treatment. The distribution has 2 independent variables ‘a = 1.5‘ and ‘b = 5’ and a user defined variable patient density to control patient flow.

The simulation logs are saved after each simulation ends and can be viewed later to investigate the effect of different factors on the optimization of ED ward facilities.

**----------------------------------------------------------------------------------------------------**

**Objectives:**

* To simulate disease assignment in an Emergency Department.
* To implement an OOP-based approach for a modular and scalable system.
* To use different probability distributions for assigning disease severity.

**----------------------------------------------------------------------------------------------------**

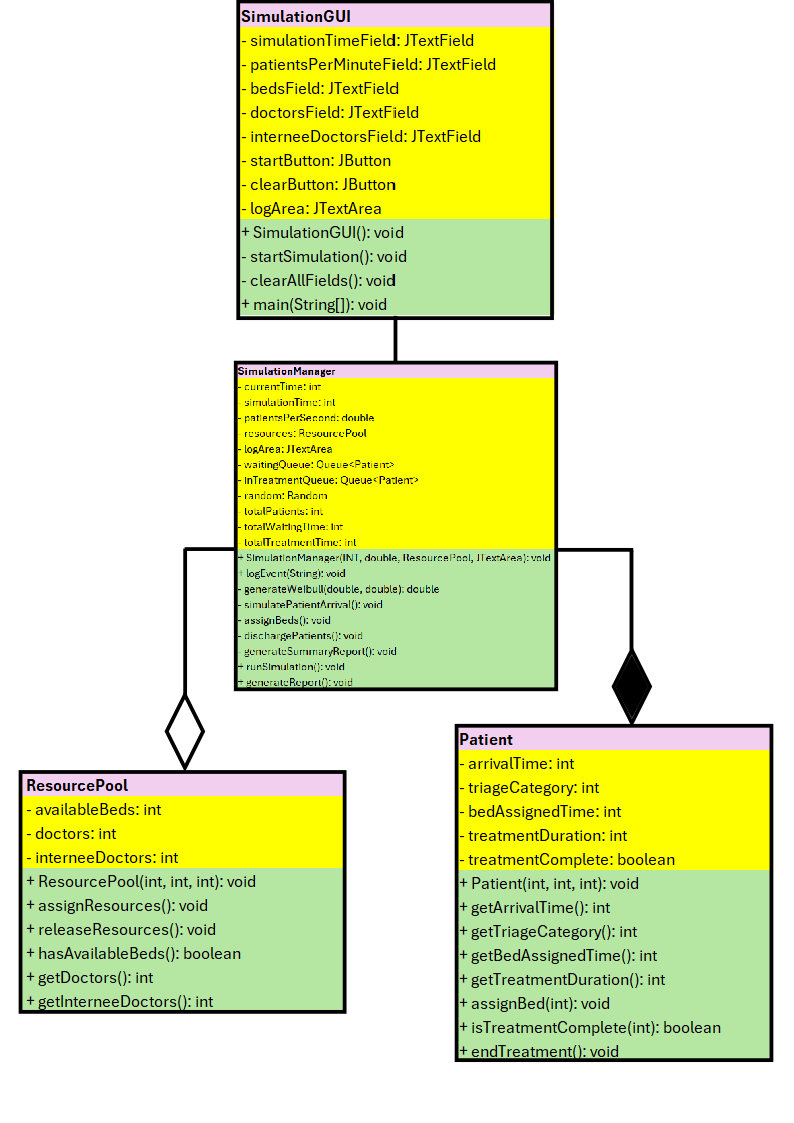
**Libraries and tools used:**

* **Hardware/Software Required:**
  + **Hardware: PC**
  + **Software: Java 8, OpenJDK 23, VS Code IDE, Eclipse IDE 2024.12**
* A clipboard and a pencil next to boxes

  Description automatically generated**Root Packages used:**
  + **javax.swing**
  + **java.awt**
  + **java.util**

**----------------------------------------------------------------------------------------------------**

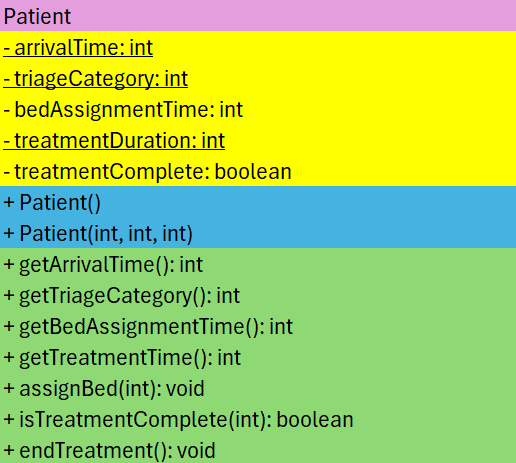
**Design overview – UML:**



**----------------------------------------------------------------------------------------------------**

**Source code overview:**

**Patient class:**

****

**Code:**

package App;

public class Patient {

    // Attributes

    private final int arrivalTime;

    private final int triageCategory;

    private int bedAssignedTime;

    private final int treatmentDuration;

    private boolean treatmentComplete;

    // Constructor

    public Patient(int arrivalTime, int triageCategory, int treatmentDuration) {

        this.arrivalTime = arrivalTime;

        this.triageCategory = triageCategory;

        this.treatmentDuration = treatmentDuration;

        this.bedAssignedTime = -1; // -1 indicates bed not assigned

        this.treatmentComplete = false;

    }

    // Getters

    public int getArrivalTime() {

        return arrivalTime;

    }

    public int getTriageCategory() {

        return triageCategory;

    }

    public int getBedAssignedTime() {

        return bedAssignedTime;

    }

    public int getTreatmentDuration() {

        return treatmentDuration;

    }

    // Assign bed and start treatment

    public void assignBed(int currentTime) {

        this.bedAssignedTime = currentTime;

    }

    // Check if treatment is complete

    public boolean isTreatmentComplete(int currentTime) {

        return currentTime - bedAssignedTime >= treatmentDuration;

    }

    // Mark treatment as complete

    public void endTreatment() {

        this.treatmentComplete = true;

    }

}

**ResourcePool class:**

**A screenshot of a computer

Description automatically generated**

**Code:**

package App;

public class ResourcePool {

    // Attributes

    private int availableBeds;

    private final int doctors;

    private final int interneeDoctors;

    // Constructor

    public ResourcePool(int availableBeds, int doctors, int interneeDoctors) {

        this.availableBeds = availableBeds;

        this.doctors = doctors;

        this.interneeDoctors = interneeDoctors;

    }

    // Assign resources

    public void assignResources() {

        if (availableBeds > 0) availableBeds--;

    }

    // Release resources

    public void releaseResources() {

        availableBeds++;

    }

    // Check if beds are available

    public boolean hasAvailableBeds() {

        return availableBeds > 0;

    }

    // Get the number of doctors

    public int getDoctors() {

        return doctors;

    }

    // Get the number of internee doctors

    public int getInterneeDoctors() {

        return interneeDoctors;

    }

}

**SimulationManager class:**

**A screenshot of a computer program

Description automatically generated**

**Code:**

package App;

import javax.swing.\*;

import java.io.FileWriter;

import java.io.IOException;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Random;

public class SimulationManager {

    // attributes

    private int currentTime;

    private final int simulationTime; // Total simulation time in seconds

    private final double patientsPerSecond;

    private final ResourcePool resources;

    private final JTextArea logArea;

    private final Queue<Patient> waitingQueue;

    private final Queue<Patient> inTreatmentQueue;

    private final Random random;

    // attributes for summary report

    private int totalPatients;

    private int totalWaitingTime;

    private int totalTreatmentTime;

    // ctor

    public SimulationManager(int simulationTime, double patientsPerSecond, ResourcePool resources, JTextArea logArea) {

        this.currentTime = 0;

        this.simulationTime = simulationTime;

        this.patientsPerSecond = patientsPerSecond;

        this.resources = resources;

        this.logArea = logArea;

        this.waitingQueue = new LinkedList<>();

        this.inTreatmentQueue = new LinkedList<>();

        this.random = new Random();

        this.totalPatients = 0;

        this.totalWaitingTime = 0;

        this.totalTreatmentTime = 0;

    }

    // Log events

    public void logEvent(String message) {

        String timeStamp = String.format("Time: %d day - ", currentTime);

        String eventLog = timeStamp + message;

        logArea.append(eventLog + "\n");

        System.out.println(eventLog);

    }

    // Generate Weibull-distributed random number

    private double generateWeibull(double shape, double scale) {

        double uniformRandom = random.nextDouble();

        return scale \* Math.pow(-Math.log(1 - uniformRandom), 1 / shape);

    }

    // Simulate patient arrival

    private void simulatePatientArrival() {

        double shape = 1.5, scale = 1.0 / patientsPerSecond;

        if (random.nextDouble() < generateWeibull(shape, scale)) {

            int triageLevel = random.nextInt(5) + 1;

            int treatmentDuration = (int) Math.ceil(generateWeibull(1.5, 5)); // Treatment time in seconds

            Patient newPatient = new Patient(currentTime, triageLevel, treatmentDuration);

            waitingQueue.add(newPatient);

            totalPatients++; // Increment patient count

            logEvent("New patient added: Triage " + triageLevel + " at time " + currentTime + " days");

        }

    }

    // Assign beds and resources to patients

    private void assignBeds() {

        while (!waitingQueue.isEmpty() && resources.hasAvailableBeds()) {

            Patient patient = waitingQueue.poll();

            resources.assignResources();

            patient.assignBed(currentTime);

            inTreatmentQueue.add(patient);

            logEvent("Patient assigned to bed: Triage " + patient.getTriageCategory() +

                    ", Arrival time " + patient.getArrivalTime());

        }

    }

    // Discharge patients after treatment

    private void dischargePatients() {

        Queue<Patient> treatedPatients = new LinkedList<>();

        for (Patient patient : inTreatmentQueue) {

            if (patient.isTreatmentComplete(currentTime)) {

                resources.releaseResources();

                patient.endTreatment();

                treatedPatients.add(patient);

                totalWaitingTime += (patient.getBedAssignedTime() - patient.getArrivalTime());

                totalTreatmentTime += patient.getTreatmentDuration();

                logEvent("Patient discharged: Triage " + patient.getTriageCategory() +

                        ", Arrival time " + patient.getArrivalTime() +

                        ", Discharge time " + currentTime + " days");

            }

        }

        inTreatmentQueue.removeAll(treatedPatients);

    }

    // Generate summary report

    private void generateSummaryReport() {

        logEvent("----------Summary Report----------");

        logEvent("Total Patients: " + totalPatients);

        logEvent("Average Waiting Time: " + (totalPatients > 0 ? (double) totalWaitingTime / totalPatients : 0) + " days");

        logEvent("Average Treatment Time: " + (totalPatients > 0 ? (double) totalTreatmentTime / totalPatients : 0) + " days");

        logEvent("Doctors to Internee Doctors Ratio: " + resources.getDoctors() + ":" + resources.getInterneeDoctors());

    }

    // Run simulation

    public void runSimulation() {

        logEvent("Simulation started...");

        while (currentTime < simulationTime) {

            simulatePatientArrival();

            dischargePatients();

            assignBeds();

            currentTime++;

        }

        logEvent("Simulation complete. Generating report...");

        generateSummaryReport(); // Add summary report

        generateReport();

    }

    // Generate final report and save it

    public void generateReport() {

    try (FileWriter writer = new FileWriter("simulation\_log.txt")) {

        writer.write(logArea.getText());

        logEvent("Report saved to simulation\_log.txt.");

    } catch (IOException e) {

        logEvent("Error saving report: " + e.getMessage());

    }

}

}

**SimulationGUI class:**

**A screenshot of a computer program

Description automatically generated**

**Code:**

package App;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class SimulationGUI extends JFrame {

    private final JTextField simulationTimeField, patientsPerMinuteField, bedsField, doctorsField, interneeDoctorsField;

    private final JButton startButton, clearButton;

    private final JTextArea logArea;

    public SimulationGUI() {

        // Set up the frame

        setTitle("Emergency Department Simulation");

        setSize(700, 500);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        getContentPane().setLayout(new BorderLayout());

        // Top panel for input fields

        JPanel inputPanel = new JPanel(new GridLayout(6, 2, 10, 10));

        inputPanel.setBackground(new Color(255, 128, 0));

        inputPanel.add(new JLabel("Simulation Time(in months): "));

        simulationTimeField = new JTextField();

        simulationTimeField.setBackground(new Color(128, 255, 0));

        inputPanel.add(simulationTimeField);

        inputPanel.add(new JLabel("Patients Density"));

        patientsPerMinuteField = new JTextField();

        patientsPerMinuteField.setBackground(new Color(128, 255, 0));

        inputPanel.add(patientsPerMinuteField);

        inputPanel.add(new JLabel("Number of Beds: "));

        bedsField = new JTextField();

        bedsField.setBackground(new Color(128, 255, 0));

        inputPanel.add(bedsField);

        inputPanel.add(new JLabel("Number of Doctors: "));

        doctorsField = new JTextField();

        doctorsField.setBackground(new Color(128, 255, 0));

        inputPanel.add(doctorsField);

        inputPanel.add(new JLabel("Number of Internee Doctors: "));

        interneeDoctorsField = new JTextField();

        interneeDoctorsField.setBackground(new Color(128, 255, 0));

        inputPanel.add(interneeDoctorsField);

        // Add "Start Simulation" and "Clear All" buttons

        startButton = new JButton("Start Simulation");

        startButton.setBackground(new Color(255, 0, 0));

        clearButton = new JButton("Clear All");

        clearButton.setBackground(new Color(255, 0, 0));

        inputPanel.add(startButton);

        inputPanel.add(clearButton);

        getContentPane().add(inputPanel, BorderLayout.NORTH);

        // Center panel for the log area

        logArea = new JTextArea();

        logArea.setBackground(new Color(128, 255, 0));

        logArea.setEditable(false);

        logArea.setFont(new Font("Monospaced", Font.BOLD, 14));

        JScrollPane scrollPane = new JScrollPane(logArea);

        getContentPane().add(scrollPane, BorderLayout.CENTER);

        // Action listeners for buttons

        startButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                startSimulation();

            }

        });

        clearButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                clearAllFields();

            }

        });

        // Make the GUI visible

        setVisible(true);

    }

    private void startSimulation() {

        try {

            // Parse user inputs

            int simulationTime = Integer.parseInt(simulationTimeField.getText()) \* 30; // Convert months to days

            int patientsPerMinute = Integer.parseInt(patientsPerMinuteField.getText());

            int beds = Integer.parseInt(bedsField.getText());

            int doctors = Integer.parseInt(doctorsField.getText());

            int interneeDoctors = Integer.parseInt(interneeDoctorsField.getText());

            // Validate inputs

            if (simulationTime <= 0 || patientsPerMinute <= 0 || beds <= 0 || doctors <= 0 || interneeDoctors <= 0) {

                JOptionPane.showMessageDialog(this, "Please enter positive numeric values!", "Input Error", JOptionPane.ERROR\_MESSAGE);

                return;

            }

            // Initialize resources

            ResourcePool resources = new ResourcePool(beds, doctors, interneeDoctors);

            // Run simulation

            SimulationManager manager = new SimulationManager(simulationTime, patientsPerMinute / 60.0, resources, logArea);

            manager.runSimulation();

            JOptionPane.showMessageDialog(this, "Simulation complete! Report saved to simulation\_log.txt.");

        } catch (NumberFormatException ex) {

            JOptionPane.showMessageDialog(this, "Please enter valid numeric values!", "Input Error", JOptionPane.ERROR\_MESSAGE);

        } catch (Exception ex) {

            JOptionPane.showMessageDialog(this, "An error occurred: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

        }

    }

    private void clearAllFields() {

        // Clear all input fields and the log area

        simulationTimeField.setText("");

        patientsPerMinuteField.setText("");

        bedsField.setText("");

        doctorsField.setText("");

        interneeDoctorsField.setText("");

        logArea.setText("");

    }

    public static void main(String[] args) {

        try {

            SwingUtilities.invokeLater(SimulationGUI::new);

        }

        catch (Exception e) {

            System.out.println("Error!");

            e.printStackTrace();

        }

    }

}

**----------------------------------------------------------------------------------------------------**

**Key features:**

* GUI interface for user friendly interaction.
* Edit different simulation parameters like number of patients, doctors, etc.
* Full simulation log and summary report.
* File handling

**------------------------------------------------------------------------------------------------**

**Primary challenges:**

* Implement the triage cycle.
* Implementation of GUI.
* File handling.
* Simulating Weibull distribution.

**------------------------------------------------------------------------------------------------**

**Functional overview - Packaging and Simulations:**

**GUI window:**

****

**Log file save:**

**A screenshot of a computer

Description automatically generated**

**Java Executable File:**

A screen shot of a computer

Description automatically generated

**----------------------------------------------------------------------------------------------------**

**Conclusion and credits:**

In making this project, all the fundamentals of the object oriented paradigm learnt throughout the semester were revitalized. I also learnt many new concepts like queues and interfaces in the Java programming language. Overall, it was a great learning experience.

**Special Thanks!**

Professor. Anum Abdul Salam,

LE Emaan Fatima

**References:**

<https://www.geeksforgeeks.org/queue-interface-java/>

<https://www.geeksforgeeks.org/file-handling-in-java/>

https://en.wikipedia.org/wiki/Emergency\_department

---------------------The END--------------------

**----------------------------------------------------------------------------------------------------**