EMPLOYEE MANAGEMENT SYSTEM – PROJECT DOCUMENTATION

1. PROJECT OVERVIEW

The Employee Management System is a Java-based console application developed to efficiently manage employee records within an organization. The system provides a structured way to store, update, retrieve, and analyze employee information while ensuring data persistence through file handling.

This application is designed using Object-Oriented Programming (OOP) principles and makes extensive use of Java Collections to handle employee data dynamically. It helps reduce manual record-keeping and improves data accuracy and accessibility.

The main purpose of the Employee Management System is to:

- Simplify employee record management
- Automate CRUD (Create, Read, Update, Delete) operations
- Enable quick searching and reporting
- Store employee data permanently for future use

2. PROJECT OBJECTIVES

The main goals of this project are:

• **Centralized Employee Data Management** – Store and manage all employee information in a single, organized system.

• **CRUD Operations** – Efficiently Create, Read, Update, and Delete employee records.

• **Search Functionality** – Quickly search employee records by ID, Name, or Department**.**

• **Data Persistence** – Ensure employee data is retained even after the program is closed, using file save and load operations.

• **Salary Calculation & Reporting** – Analyze employee salaries and generate reports including total, average, highest, and lowest salary.

• **Department-wise Summary** – Provide department-level insights, such as number of employees and average salary per department.

• **Robust Exception Handling** – Gracefully handle file errors, invalid inputs, or data inconsistencies without crashing the program.

• **Menu-driven User Interface** – Provide an interactive console-based interface for users to perform operations easily.

• **Efficient Lookup** – Use **HashMap** for fast access and retrieval of employee records by their ID.

• **Flexible Data Storage** – Store employee data in text (**.**txt) files using serialization or file I/O operations.

• **Beginner-friendly & Extendable** – Simple, easy-to-understand design that can be extended in the future with additional features like performance tracking or leave management.

3. SETUP & INSTALLATION INSTRUCTIONS

1. Install Java Development Kit (JDK) on the system
2. Download JDK from the official Oracle or OpenJDK website
3. Verify Java installation by running the command: java -version in the command prompt
4. Install a Java-supported IDE such as Eclipse or Visual Studio Code
5. Open the IDE and create a new Java project
6. Name the project as EmployeeManagementSystem
7. Create a package named as INTERNSHIP
8. Inside the package, create the following Java class files:
    - Employee.java
    - EmployeeManagementSystem.java
    - EmployeeFileHandler.java
    - EmployeeReportGenerator.java
9. Copy the respective source code into each file and save them
10. Compile the project to ensure there are no errors
11. Run the EmployeeManagementSystem.java file which contains the main() method
12. The menu-driven Employee Management System will start in the console, allowing you to add, view, search, update, delete employees, generate reports, and save/load data

4. CODE STRUCTURE EXPLANATION

EMPLOYEEFILEHANDLER.JAVA (FILE HANDLING)

- This class is responsible for saving and loading employee data from a text file.
- It uses BufferedWriter to write employee details line by line into employees.txt.
- Each employee is converted into a comma-separated string using toFileString().
- It uses BufferedReader to read the file and recreate Employee objects.
- Exception handling ensures the program doesn’t crash if the file is missing.

```java
package Employee_Management_System;

import java.io.*;  // For file reading/writing

import java.util.ArrayList;  // For the employee list

public class EmployeeFileHandler {  // Class for file operations
   private static final String FILE_NAME = "employees.text";  // File name constant

   // Saves employees to file
   public void saveEmployees(ArrayList<Employee> employees) {
      try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_NAME))) {  // Opens file for writing
         for (Employee emp : employees) {  // Loops through employees
            writer.write(emp.toFileString());  // Writes serialized string
            writer.newLine();  // Adds newline
         }
         System.out.println("Data saved.");  // Confirms save
      } catch (IOException e) {  // Catches I/O errors
         System.out.println("Error saving: " + e.getMessage());  // Prints error
      }
   }

   // Loads employees from file
   public ArrayList<Employee> loadEmployees() {
      ArrayList<Employee> employees = new ArrayList<>();  // Creates empty list
      try (BufferedReader reader = new BufferedReader(new FileReader(FILE_NAME))) {  // Opens file for reading
         String line;
         while ((line = reader.readLine()) != null) {  // Reads each line
            employees.add(Employee.fromFileString(line));  // Deserializes and adds to list
         }
         System.out.println("Data loaded.");  // Confirms load
      } catch (IOException e) {  // Catches I/O errors
         System.out.println("No data file or error loading. Starting fresh.");  // Prints message
      }
      return employees;  // Returns the list
   }
}
```

EMPLOYEE.JAVA (MODEL / ENTITY CLASS)

- This class represents a **single employee** with id, name, department, position, salary, and join date.
- It uses **LocalDate** to store joining date in a standard format.
- Getter and setter methods allow **safe access and modification** of data.
- toFileString() converts employee data into a **storable text format**.

- fromFileString() recreates an Employee object from file data.

```
package Employee_Management_System;
import java.time.LocalDate;  // Imports LocalDate for handling dates (e.g., join date)

public class Employee {  // Defines the Employee class
    private int id;  // Unique identifier for the employee
    private String name;  // Employee's name
    private String department;  // Department they work in
    private String position;  // Job position
    private double salary;  // Salary amount
    private LocalDate joinDate;  // Date they joined (using LocalDate for date handling)

    // Constructor to create an Employee object with all attributes
    public Employee(int id, String name, String department, String position, double salary, LocalDate
joinDate) {
        this.id = id;  // Assigns the ID
        this.name = name;  // Assigns the name
        this.department = department;  // Assigns the department
        this.position = position;  // Assigns the position
        this.salary = salary;  // Assigns the salary
        this.joinDate = joinDate;  // Assigns the join date
    }

    // Getter methods to retrieve attribute values
    public int getId() { return id; }  // Returns the ID
    public String getName() { return name; }  // Returns the name
    public String getDepartment() { return department; }  // Returns the department
    public String getPosition() { return position; }  // Returns the position
    public double getSalary() { return salary; }  // Returns the salary
    public LocalDate getJoinDate() { return joinDate; }  // Returns the join date

    // Setter methods to update attribute values (used in updates)
    public void setName(String name) { this.name = name; }  // Updates the name
    public void setDepartment(String department) { this.department = department; }  // Updates the
department
    public void setPosition(String position) { this.position = position; }  // Updates the position
    public void setSalary(double salary) { this.salary = salary; }  // Updates the salary
    public void setJoinDate(LocalDate joinDate) { this.joinDate = joinDate; }  // Updates the join date

    // toString method for displaying employee details in a readable format
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Dept: " + department + ", Pos: " + position + ", Salary:
" + salary + ", Joined: " + joinDate;
        // Returns a formatted string with all details
    }
```

```
    // Method to serialize (convert) the employee to a comma-separated string for file saving
    public String toFileString() {
        return id + "," + name + "," + department + "," + position + "," + salary + "," + joinDate;
        // Combines attributes into a single string, separated by commas
    }

    // Static method to deserialize (convert back) a comma-separated string from file into an
Employee object
    public static Employee fromFileString(String line) {
        String[] parts = line.split(",");  // Splits the line by commas into an array
        return new Employee(Integer.parseInt(parts[0]), parts[1], parts[2], parts[3],
Double.parseDouble(parts[4]), LocalDate.parse(parts[5]));
        // Parses each part and creates a new Employee object
    }
}
```

EMPLOYEEMANAGEMENTSYSTEM.JAVA (MAIN LOGIC CLASS)

- This is the core class that manages employees using ArrayList and HashMap.
- ArrayList stores all employees, while HashMap allows fast lookup by ID.
- It implements CRUD operations: add, view, update, and delete employees.
- It provides search features by name and department.
- A menu-driven main method allows users to interact with the system.

```
package Employee_Management_System;

import java.time.LocalDate;  // For date handling

import java.util.*;  // For collections like ArrayList and HashMap



public class EmployeeManagementSystem {  // Main class for the system

    private ArrayList<Employee> employees = new ArrayList<>();  // List to store all employees

    private HashMap<Integer, Employee> employeeMap = new HashMap<>();  // Map for quick ID-
based lookup

    private int nextId = 1;  // Auto-incrementing ID for new employees

    private EmployeeFileHandler fileHandler = new EmployeeFileHandler();  // Instance for file
operations

    private EmployeeReportGenerator reportGenerator = new EmployeeReportGenerator(); //
Instance for reports
```

```
    // Constructor: Loads data from file on startup
    public EmployeeManagementSystem() {
        loadData(); // load employees first
    }
private void loadData() {
        ArrayList<Employee> loaded = fileHandler.loadEmployees();
       employees.clear();
        employeeMap.clear();
        nextId = 1;
   for (Employee emp : loaded) {
            employees.add(emp);
            employeeMap.put(emp.getId(), emp);
            if (emp.getId() >= nextId) {
                nextId = emp.getId() + 1;
            }
        }
    }

    // Saves employees to file
    private void saveData() {
        fileHandler.saveEmployees(employees);  // Delegates to file handler
    }
  // CRUD: Adds a new employee
```

```
   public void addEmployee(String name, String department, String position, double salary,
LocalDate joinDate) {

       Employee emp = new Employee(nextId++, name, department, position, salary, joinDate);  //
Creates new Employee

       employees.add(emp);  // Adds to list

       employeeMap.put(emp.getId(), emp);  // Adds to map

       saveData();

   }

 // CRUD: Displays all employees

   public void viewAllEmployees() {

       if (employees.isEmpty()) {

           System.out.println("No employees.");

           return;

       }

      System.out.println("=== ALL EMPLOYEES ===");

       System.out.println("ID | Name       | Department   | Position   | Salary     | Join Date");

       System.out.println("-----------------------------------------------------------------");

      for (Employee e : employees) {

           System.out.println(

               "E" + e.getId() + " | " +

               e.getName() + " | " +

               e.getDepartment() + " | " +

               e.getPosition() + " | " +

               "₹" + e.getSalary() + " | " +

               e.getJoinDate()
```

```
            );
        }
    }
// CRUD: Displays employee by ID
    public void viewEmployeeById(int id) {
        Employee e = employeeMap.get(id);  // Lookup employee in map

        if (e == null) {
            System.out.println("Employee with ID E" + id + " not found.");
            return;
        }

        // Print table header
        System.out.println("=== EMPLOYEE DETAILS ===");
        System.out.println("ID | Name      | Department   | Position   | Salary     | Join Date");
        System.out.println("-----------------------------------------------------------------");

        // Print the employee details
        System.out.println(
            "E" + e.getId() + " | " +
            e.getName() + " | " +
            e.getDepartment() + " | " +
            e.getPosition() + " | " +
            "₹" + e.getSalary() + " | " +
```

```
            e.getJoinDate()

        );

    }

// CRUD: Updates an employee

    public void updateEmployee(int id, String name, String department, String position, double
salary, LocalDate joinDate) {

        Employee emp = employeeMap.get(id);  // Finds employee

        if (emp == null) {  // If not found

            System.out.println("Not found.");  // Prints message

            return;  // Exits

        }

        if (name != null && !name.isEmpty()) emp.setName(name);  // Updates name if provided

        if (department != null && !department.isEmpty()) emp.setDepartment(department);  //
Updates department

        if (position != null && !position.isEmpty()) emp.setPosition(position);  // Updates position

        if (salary >= 0) emp.setSalary(salary);  // Updates salary if valid

        if (joinDate != null) emp.setJoinDate(joinDate);  // Updates date if provided

        saveData();

    }



    // CRUD: Deletes an employee

    public void deleteEmployee(int id) {

        Employee emp = employeeMap.remove(id);  // Removes from map

        if (emp != null) {  // If found

            employees.remove(emp);// Removes from list
```

```
        saveData();

    } else {
        System.out.println("Not found.");  // If not found
    }
}

public void searchByName(String name) {
    ArrayList<Employee> results = new ArrayList<>();

    for (Employee e : employees) {
        if (e.getName().toLowerCase().contains(name.toLowerCase())) {
            results.add(e);
        }
    }

    if (results.isEmpty()) {
        System.out.println("No employees found with name: " + name);
        return;
    }

    System.out.println("=== SEARCH RESULTS BY NAME ===");
    System.out.println("ID | Name       | Department   | Position   | Salary     | Join Date");
    System.out.println("-----------------------------------------------------------------");
```

```
    for (Employee e : results) {
        System.out.println(
            "E" + e.getId() + " | " +
            e.getName() + " | " +
            e.getDepartment() + " | " +
            e.getPosition() + " | " +
            "₹" + e.getSalary() + " | " +
            e.getJoinDate()
        );
    }
}


  public void searchByDepartment(String department) {
     ArrayList<Employee> results = new ArrayList<>();
    for (Employee e : employees) {
        if (e.getDepartment().toLowerCase().contains(department.toLowerCase())) {
            results.add(e);
        }
    }
if (results.isEmpty()) {
        System.out.println("No employees found in department: " + department);
        return;
```

```
    }
    System.out.println("=== SEARCH RESULTS BY DEPARTMENT ===");
    System.out.println("ID | Name      | Department   | Position   | Salary     | Join Date");
    System.out.println("-----------------------------------------------------------------");

    for (Employee e : results) {
        System.out.println(
            "E" + e.getId() + " | " +
            e.getName() + " | " +
            e.getDepartment() + " | " +
            e.getPosition() + " | " +
            "₹" + e.getSalary() + " | " +
            e.getJoinDate()
        );
    }
  }


// Generates salary report
  public void generateSalaryReport() {
    reportGenerator.generateSalaryReport(employees);  // Delegates to report generator
  }

  // Generates department report
```

```
    public void generateDepartmentReport() {
        reportGenerator.generateDepartmentReport(employees);  // Delegates to report generator
    }

    // Main method: Runs the menu-driven interface
    public static void main(String[] args) {
        EmployeeManagementSystem system = new EmployeeManagementSystem();  // Creates system instance
        Scanner sc = new Scanner(System.in);  // For user input
        while (true) {  // Infinite loop for menu
            System.out.println("\n1. Add \n2. View All \n3. View by ID \n4. Update \n5. Delete \n6. Search Name \n7. Search Dept \n8. Salary Report \n9. Dept Report \n10. Save & Exit");
            System.out.print("Choice: ");
            int choice = Integer.parseInt(sc.nextLine());  // Reads user choice
            try {
                switch (choice) {  // Handles menu options
                    case 1:  // Add employee
                        System.out.print("Name: ");
                        String name = sc.nextLine();
                        System.out.print("Department: ");
                        String dept = sc.nextLine();
                        System.out.print("Position: ");
                        String pos = sc.nextLine();
                        System.out.print("Salary: ");
                        double sal = Double.parseDouble(sc.nextLine());
```

```
    System.out.print("Join Date (YYYY-MM-DD): ");
    LocalDate date = LocalDate.parse(sc.nextLine());
    system.addEmployee(name, dept, pos, sal, date);// Calls add method
    System.out.println("Employee Added Successfully!!!!");
    break;
case 2: system.viewAllEmployees(); break;  // View all
case 3:  // View by ID
    System.out.print("ID: ");
    int id = Integer.parseInt(sc.nextLine());
    system.viewEmployeeById(id);
    break;
case 4:  // Update
    System.out.print("ID: ");
    id = Integer.parseInt(sc.nextLine());
    System.out.print("New Name (blank to skip): ");
    name = sc.nextLine();
    System.out.print("New Department (blank to skip): ");
    dept = sc.nextLine();
    System.out.print("New Position (blank to skip): ");
    pos = sc.nextLine();
    System.out.print("New Salary (-1 to skip): ");
    sal = Double.parseDouble(sc.nextLine());
    System.out.print("New Join Date (blank to skip): ");
    String dateStr = sc.nextLine();
```

```
    LocalDate newDate = dateStr.isEmpty() ? null : LocalDate.parse(dateStr);
    system.updateEmployee(id, name, dept, pos, sal, newDate);
    System.out.println("Employee Updated Successfully!!!!");
    break;
case 5:  // Delete
    System.out.print("ID: ");
    id = Integer.parseInt(sc.nextLine());
    system.deleteEmployee(id);
    System.out.println("Employee Deleted Successfully!!!!");
    break;
case 6:  // Search by name
    System.out.print("Name: ");
    name = sc.nextLine();
    system.searchByName(name);
    break;
case 7:  // Search by dept
    System.out.print("Department: ");
    dept = sc.nextLine();
    system.searchByDepartment(dept);
    break;
case 8: system.generateSalaryReport(); break;  // Salary report
case 9: system.generateDepartmentReport(); break;  // Dept report
case 10: system.saveData(); return;  // Save and exit
default: System.out.println("Invalid.");  // Invalid choice
```

```
            }
        } catch (Exception e) {  // Catches any errors
            System.out.println("Error: " + e.getMessage());  // Prints error
        }

    }
  }
}
```

EMPLOYEEREPORTGENERATOR.JAVA (REPORTS & ANALYSIS)

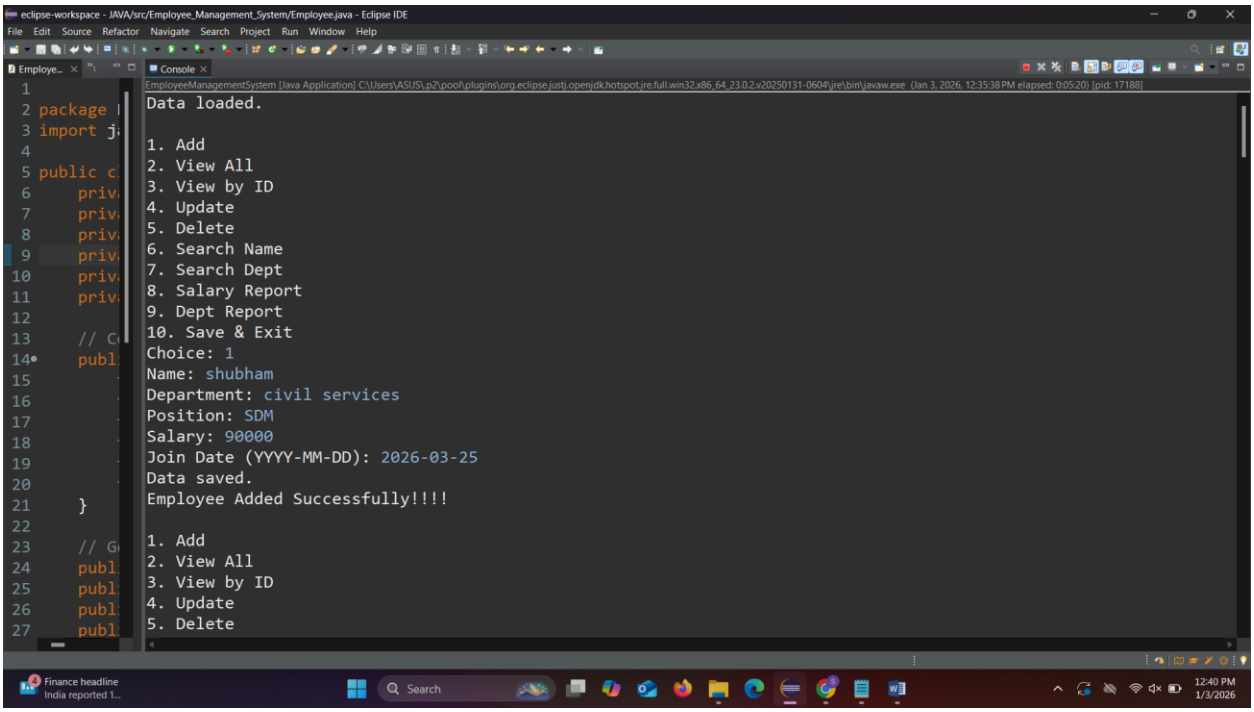
- This class is used to generate reports from employee data.
- It calculates total and average salary using Java Streams.
- Employees are grouped department-wise using a HashMap.
- It displays employees under each department clearly.
- Keeps reporting logic separate from main system logic (good design).

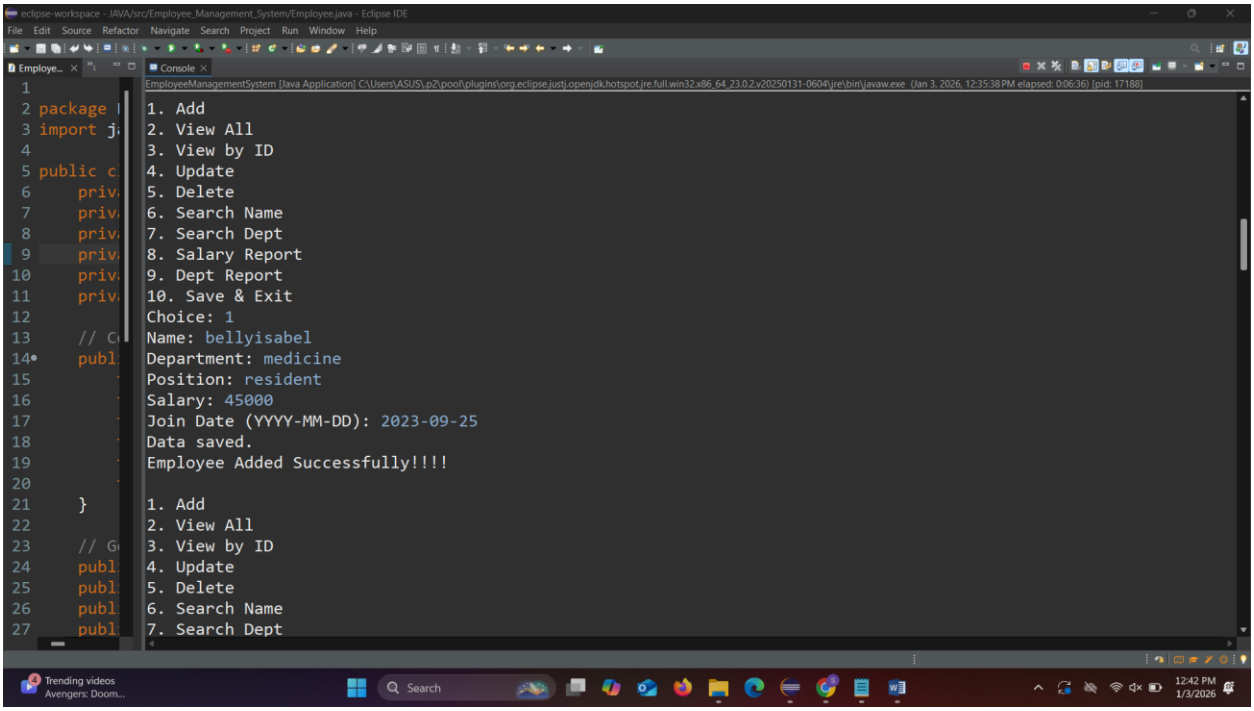
```
package Employee_Management_System;


import java.util.ArrayList;
import java.util.HashMap;

public class EmployeeReportGenerator {

   // SALARY REPORT
   public void generateSalaryReport(ArrayList<Employee> employees) {

      if (employees == null || employees.isEmpty()) {
         System.out.println("No employees to generate salary report.");
         return;
      }

      double totalSalary = 0;
      for (Employee e : employees) {
         totalSalary += e.getSalary();
      }

      double avgSalary = totalSalary / employees.size();
```

```
    System.out.println("\n=== SALARY REPORT ===");
    System.out.println("Total Employees : " + employees.size());
    System.out.println("Total Salary    : ₹" + totalSalary);
    System.out.println("Average Salary  : ₹" + avgSalary);
}

// DEPARTMENT REPORT
public void generateDepartmentReport(ArrayList<Employee> employees) {

    if (employees == null || employees.isEmpty()) {
        System.out.println("No employees to generate department report.");
        return;
    }

    HashMap<String, ArrayList<Employee>> deptMap = new HashMap<>();

    for (Employee e : employees) {
        deptMap.computeIfAbsent(e.getDepartment(), k -> new ArrayList<>()).add(e);
    }

    System.out.println("\n=== DEPARTMENT REPORT ===");

    for (String dept : deptMap.keySet()) {
        System.out.println("\nDepartment: " + dept);
        System.out.println("-----------------------------------");

        for (Employee e : deptMap.get(dept)) {
            System.out.println(
                    "ID: E" + e.getId() +
                    ", Name: " + e.getName() +
                    ", Position: " + e.getPosition() +
                    ", Salary: ₹" + e.getSalary() +
                    ", Join Date: " + e.getJoinDate()
            );
        }
    }
```

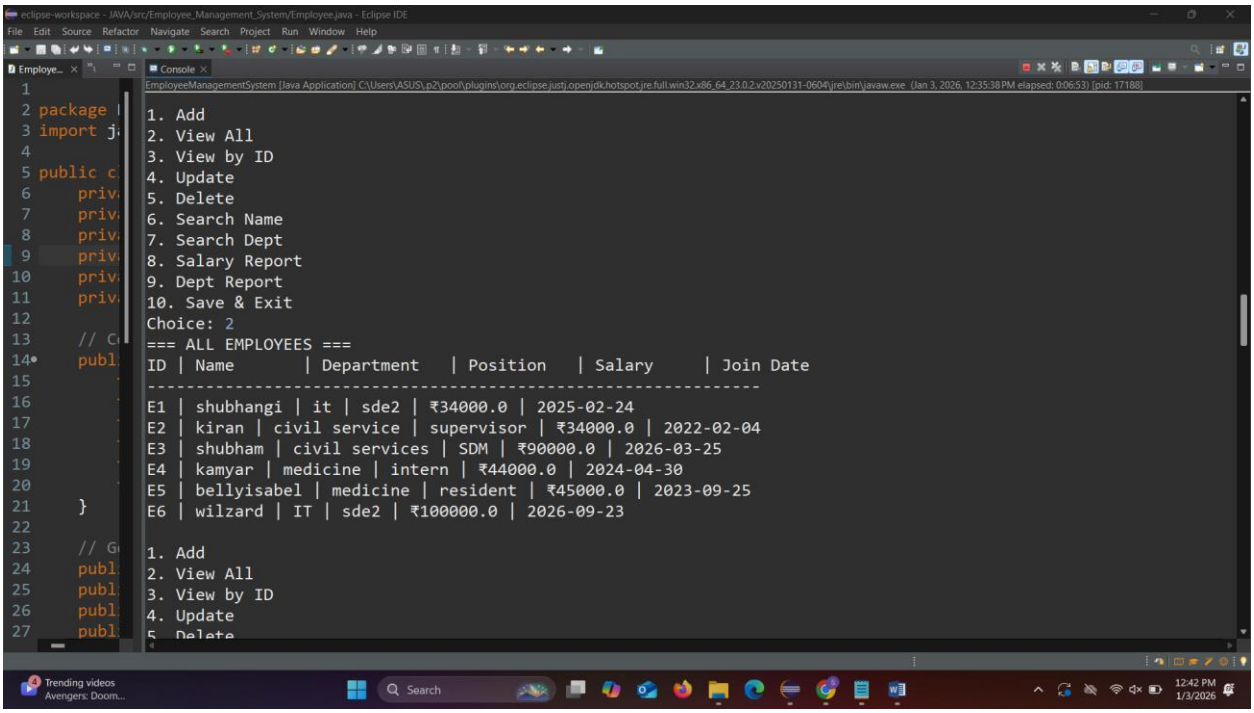
5. SCREENSHOTS OF WORKING APPLICATION

ADD EMPLOYEE

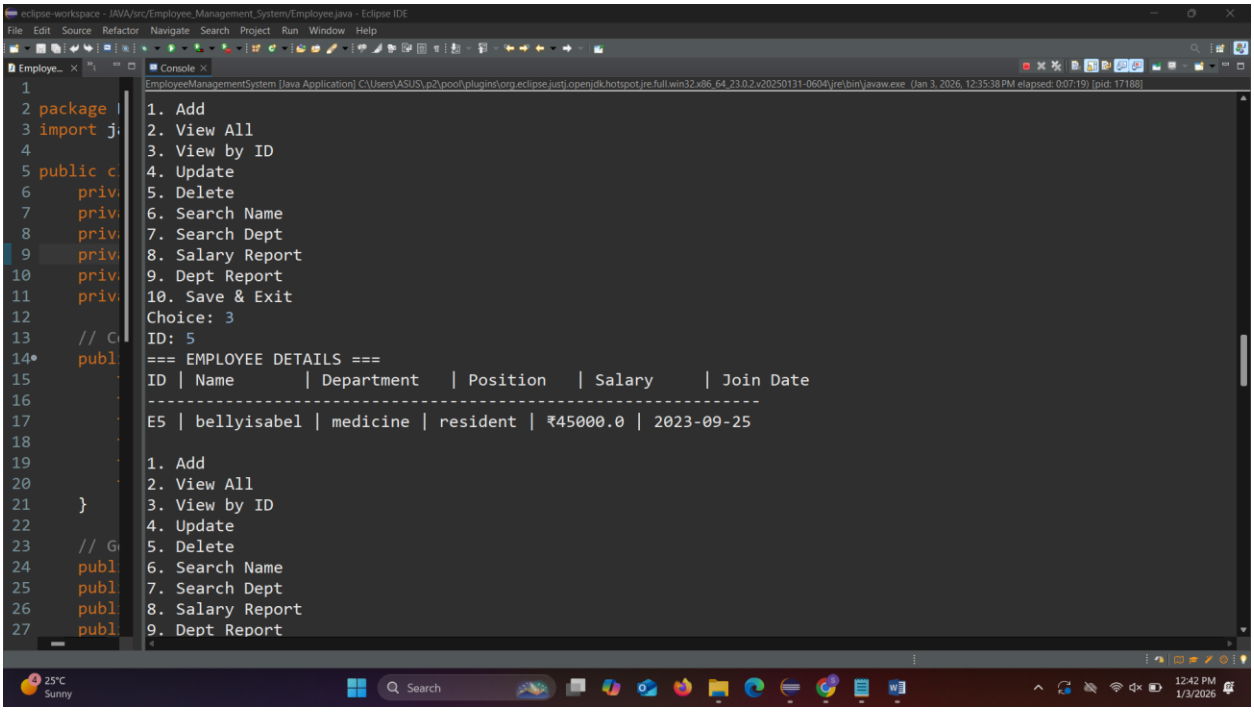




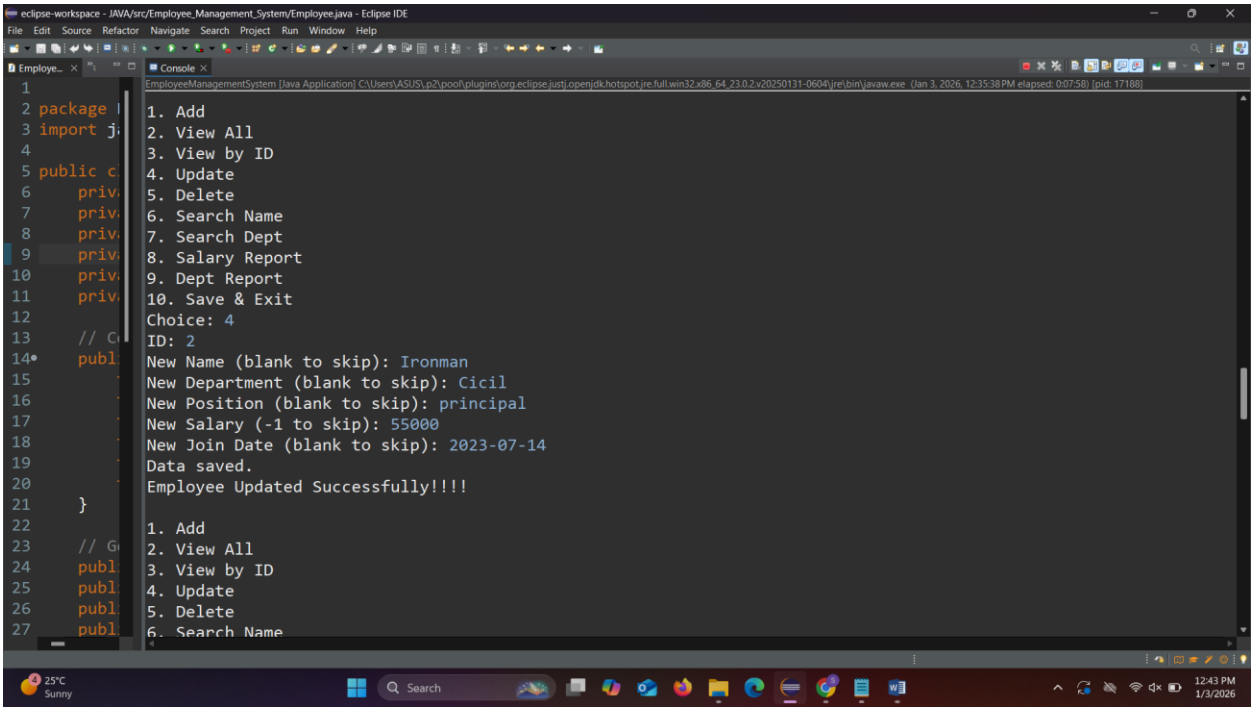
VIEW ALL EMPLOYEES



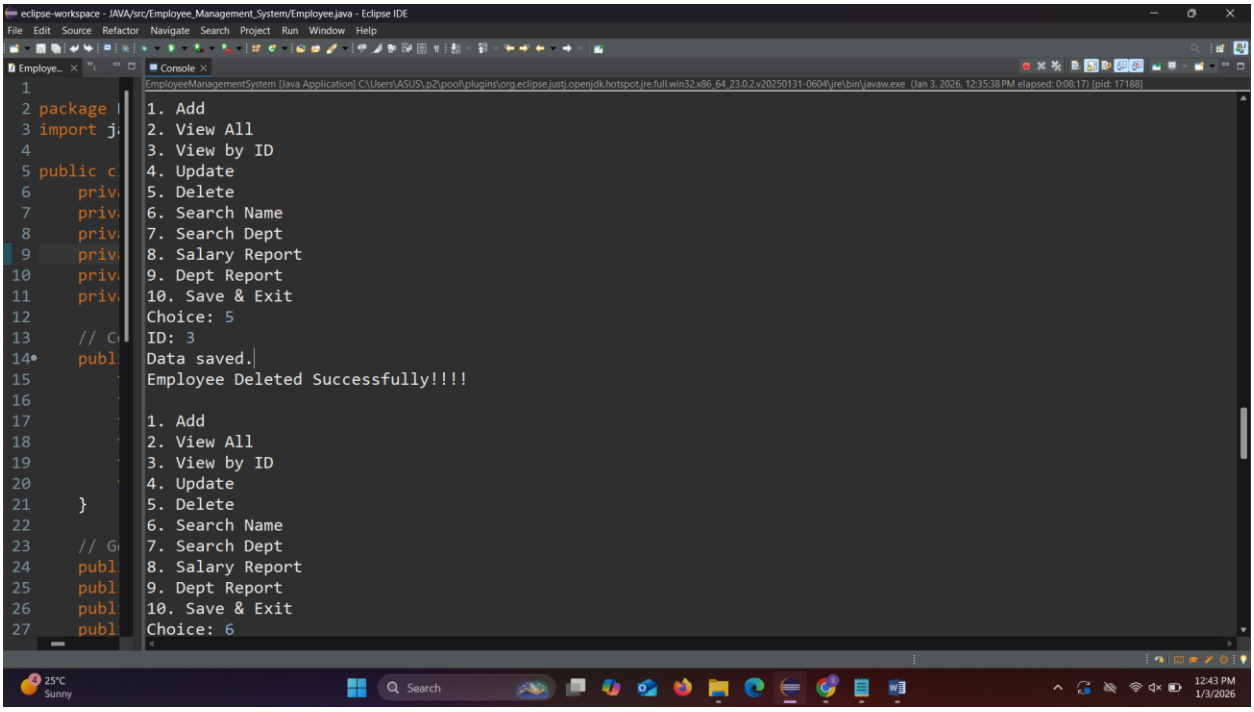
VIEW EMPLOYEE BY ID



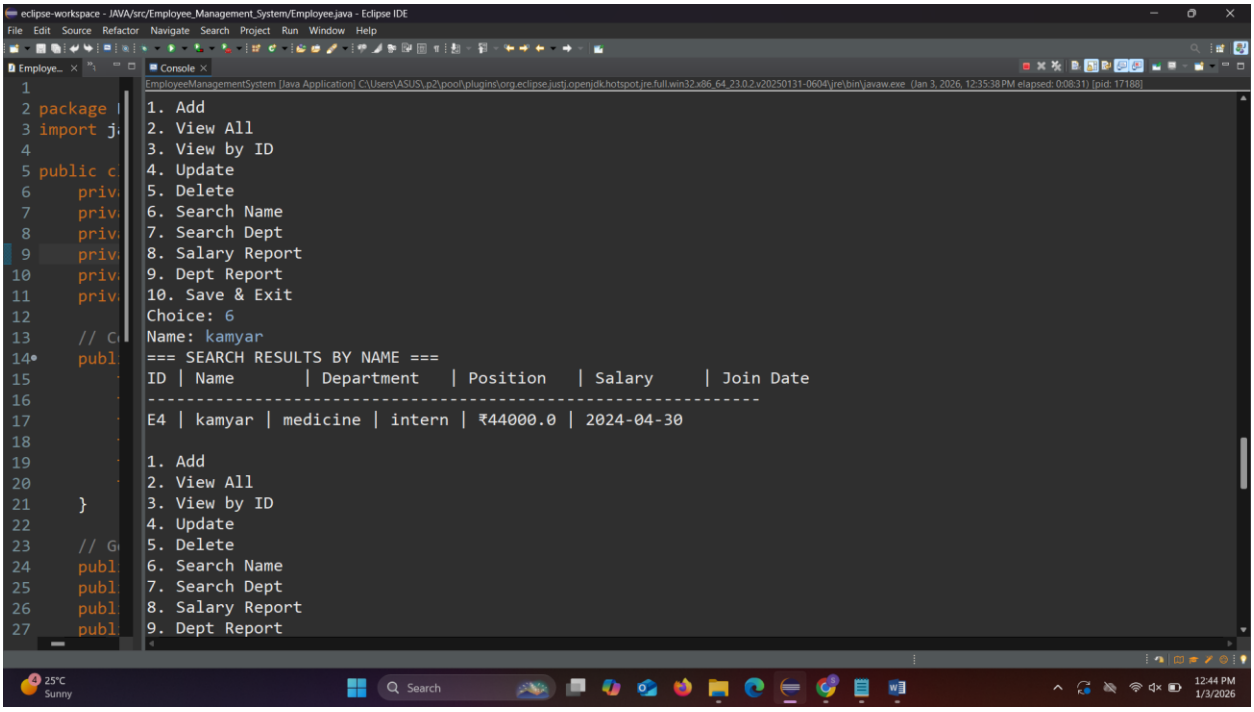
UPDATE EMPLOYEE INFORMATION



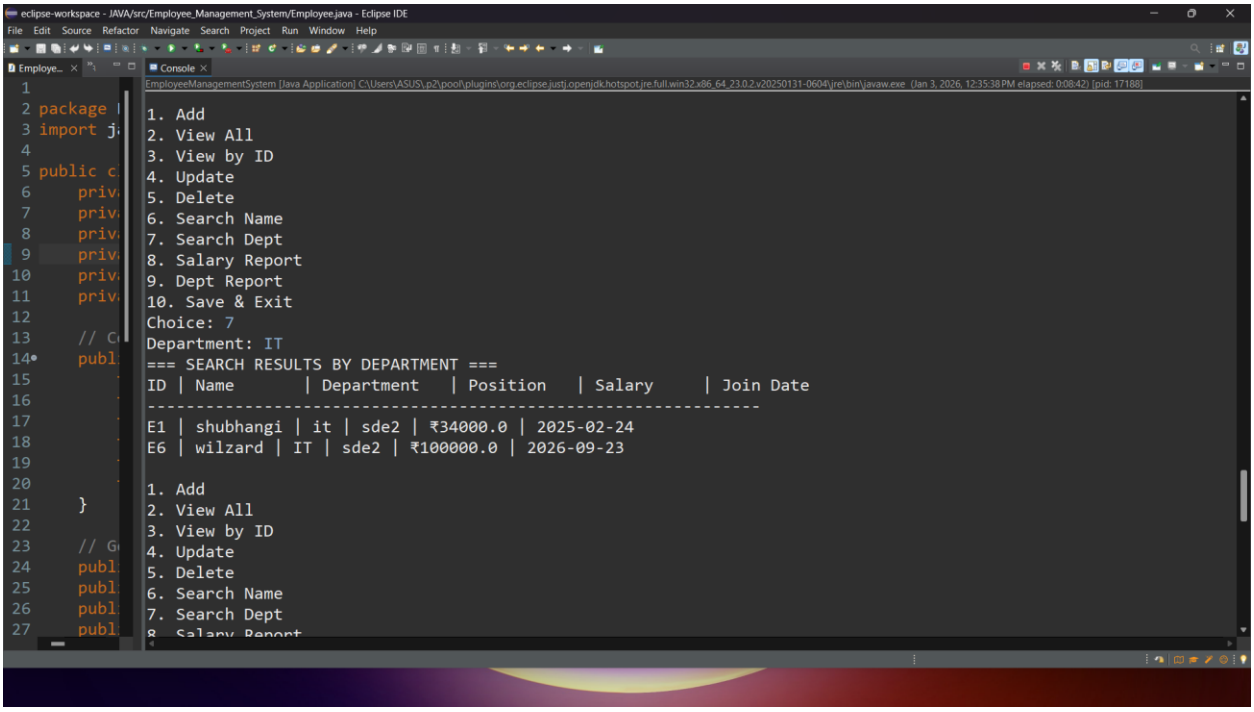
DELETE EMPLOYEE



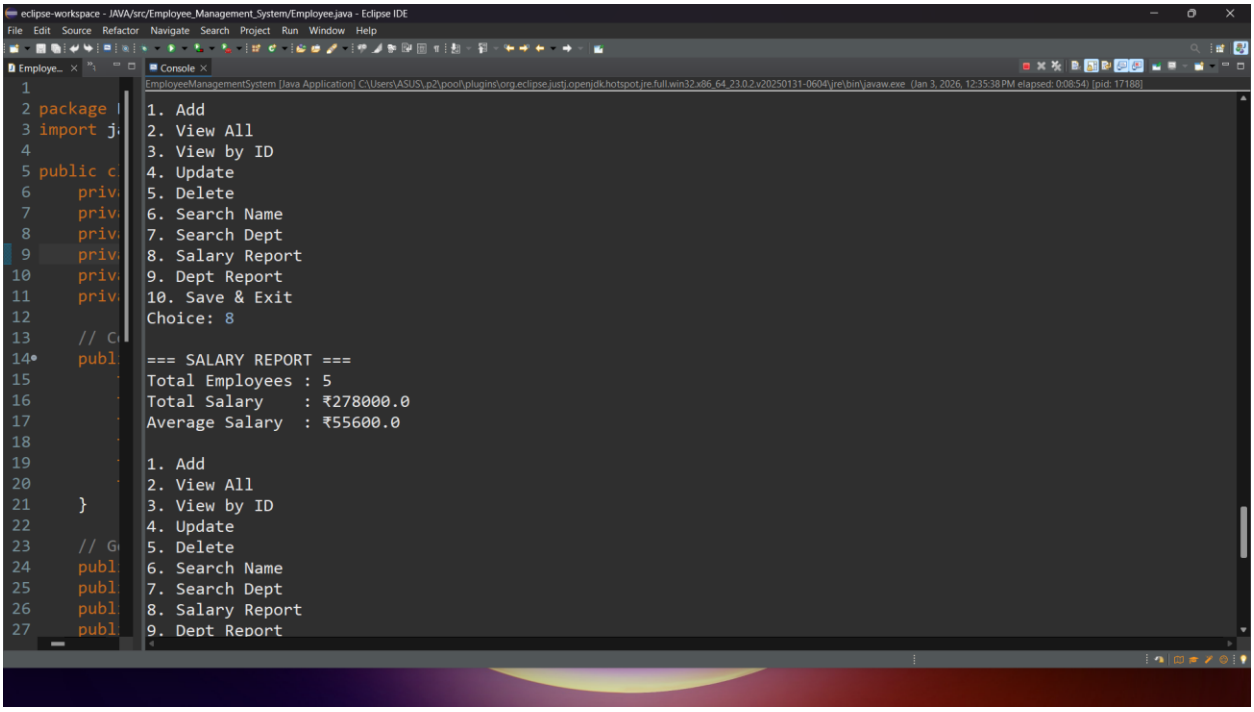
SEARCH EMPLOYEE BY NAME



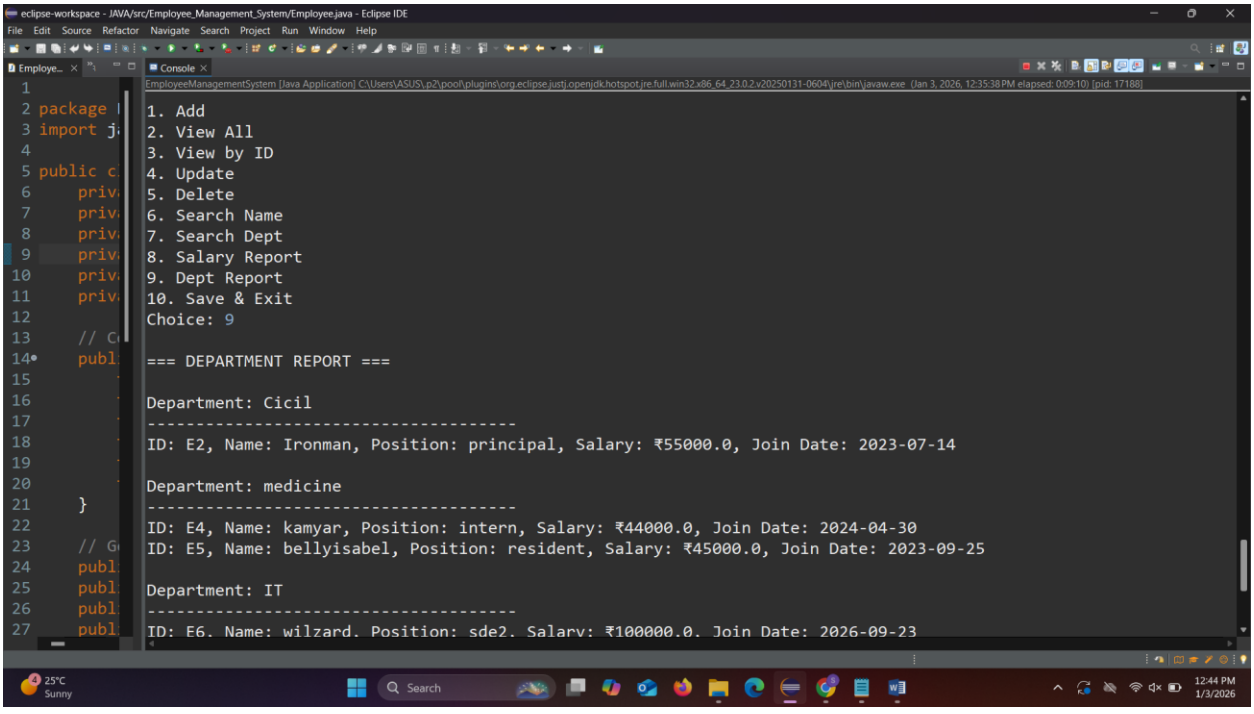
SEARCH EMPLOYEE BY DEPARTMENT

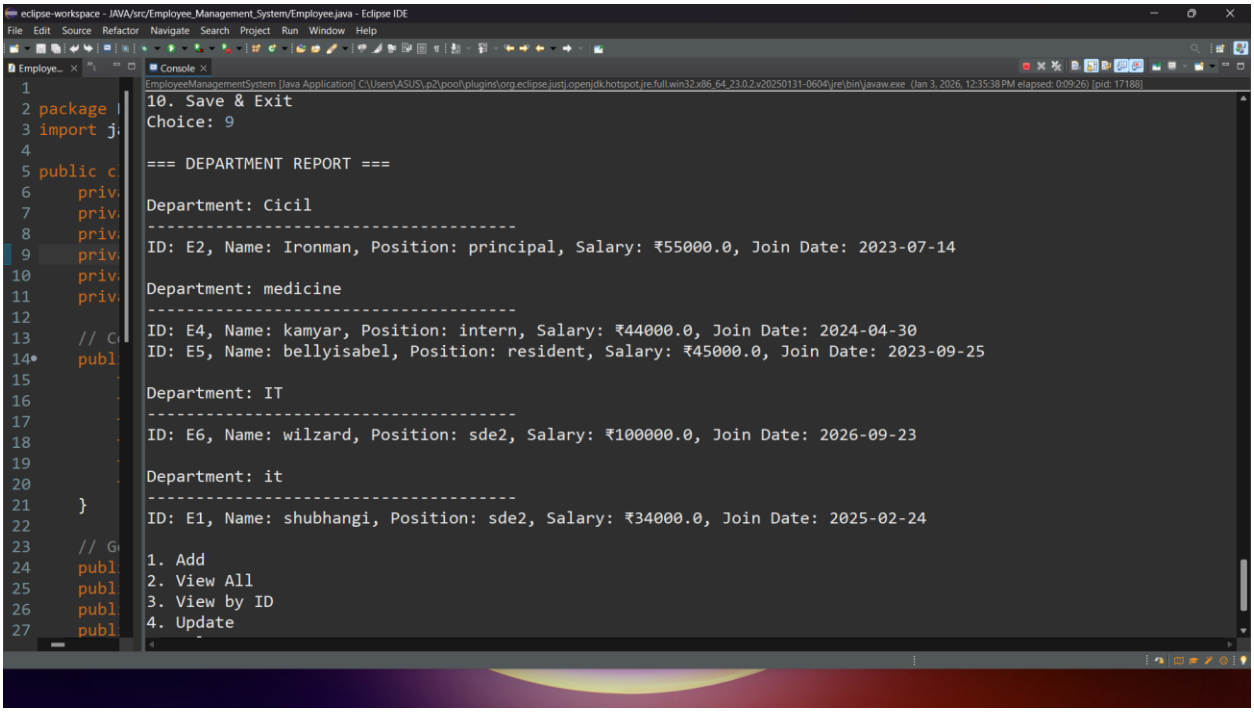


SALARY REPORT



DEPARTMENT REPORT





6. SAMPLE EXAMPLE

```
Name: KANAKESHWARI
Department: IT
Position: SOFTWARE ENGINEER
Salary: 50000
Join Date (YYYY-MM-DD): 2023-08-26
Data saved.
Employee Added Successfully!!!!
Name: shifanaaz
Department: IT
Position: Application Engineer
Salary: 58000
Join Date (YYYY-MM-DD): 2024-06-08
Data saved.
Employee Added Successfully!!!!
```

7. HOW THE PROJECT MEETS TECHNICAL REQUIREMENTS

1. Create Employee class with attributes

Requirement:
id, name, department, position, salary, joinDate

Fulfillment:

- The Employee class defines all required attributes as private variables.
- Uses LocalDate joinDate for proper date handling.
- Constructor initializes all fields.
- Getter and setter methods provide controlled access.

2. Use ArrayList to store employee objects

Fulfillment:

- ArrayList<Employee> employees is used in EmployeeManagementSystem.
- Stores all employee objects in memory.
- Allows dynamic addition, removal, and traversal.

```
private ArrayList<Employee> employees = new ArrayList<>();
```

3. Use HashMap for quick employee lookup by ID

Fulfillment:

- HashMap<Integer, Employee> stores employees with ID as key.
- Provides O(1) time complexity for search by ID.
- Used in view, update, and delete operations.

```
private HashMap<Integer, Employee> employeeMap = new HashMap<>();
```

4. Implement CRUD operations

- addEmployee() creates and stores a new employee.

- viewAllEmployees() displays all employees.
- viewEmployeeById() retrieves employee using HashMap.

- updateEmployee() modifies existing employee data.

- deleteEmployee() removes employee from both ArrayList and HashMap.

5. Add file persistence (Save & Load)

Fulfillment:

- EmployeeFileHandler class handles file operations.
- Employee data is saved to employees.txt.
- Data is loaded when application starts.

```
fileHandler.saveEmployees(employees);
employees = fileHandler.loadEmployees();
```

6. Implement comprehensive exception handling

Fulfillment:

- try-catch blocks handle file I/O errors.
- Prevents application crash on invalid input.
- Displays user-friendly error messages.

```
catch (IOException e) {
   System.out.println("Error saving data");
}
```

7. Create search functionality (Name / Department)

Fulfillment:

- Uses Java Streams to search employees.
- Case-insensitive search improves usability.

```
searchByName()
searchByDepartment()
```

8. Add salary calculation & reporting features

Fulfillment:

- EmployeeReportGenerator calculates: 1. Total salary 2. Average salary
- Generates department-wise employee report.

```
generateSalaryReport()
generateDepartmentReport()
```

9 . CONCLUSION

This Employee Management System fulfills all technical requirements by using collections for storage, HashMap for fast lookup, CRUD operations for management, file handling for persistence, exception handling for stability, search features for usability, and reporting for salary analysis.