

Data structure

# Employee record management system

Using Linked list

Safanh alzahrani 2210958

Danah alsubaie 2210611

## Introduction:

The Employee Record Management System using linked list is designed to address the need for efficient management of employee records. This project aims to provide a comprehensive solution for organizing, updating, and accessing employee information. By utilizing linked lists, the system offers functionalities such as inserting, deleting, updating, and searching employee records. Additionally, it includes a feature to automatically adjust salaries based on work hours. The project also proposes the development of a Graphical User Interface (GUI) to enhance user experience. Notably, in implementing functionalities such as searching for an employee, displaying all employee information, and updating specific employee details, action listeners methods in different classes are employed to ensure seamless interaction and response to user inputs. Through this system, organizations can streamline their record-keeping processes and ensure accurate and accessible employee data management.

## Employee class (Node class):

```
public class Employee {  
  
    private String name;  
    private String empId;  
    private String firstDayOfWork;  
    private String phoneNumber;  
    private String address;  
    private int workHours;  
    private double salary;  
  
    private Employee next;  
  
    public Employee(String name, String empId, String firstDayOfWork, String phoneNumber, String address, int workHours, double salary) {  
        this.name = name;  
        this.empId = empId;  
        this.firstDayOfWork = firstDayOfWork;  
        this.phoneNumber = phoneNumber;  
        this.address = address;  
        this.workHours = workHours;  
        this.salary = salary;  
        this.next = null;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

```

    public void setEmpId(String empId) {
        this.empId = empId;
    }

    public String getFirstDayOfWork() {
        return firstDayOfWork;
    }

    public void setFirstDayOfWork(String firstDayOfWork) {
        this.firstDayOfWork = firstDayOfWork;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public int getWorkHours() {
        return workHours;
    }

    public void setWorkHours(int workHours) {
        this.workHours = workHours;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

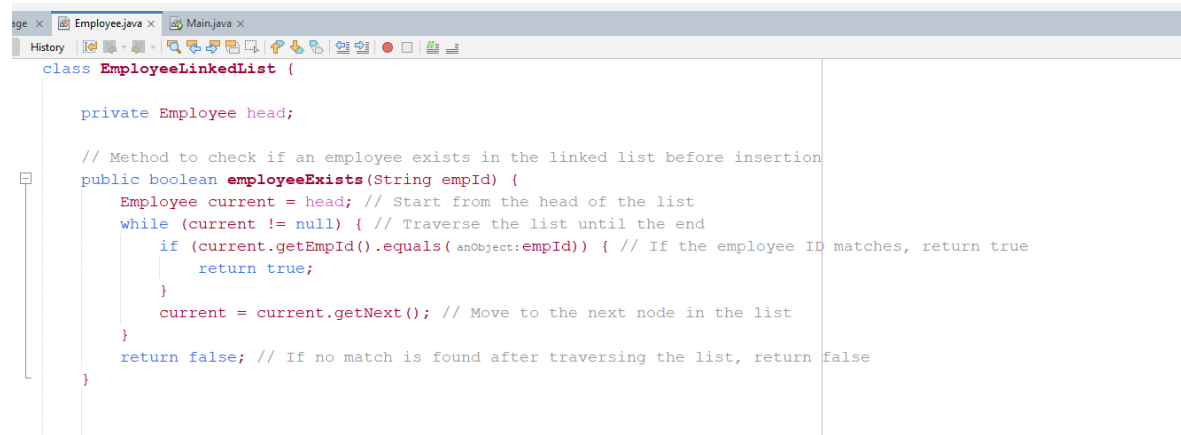
    public Employee getNext() {
        return next;
    }

    public void setNext(Employee next) {
        this.next = next;
    }
}

```

## EmployeeLinkedList class (linked list class):

### 1-employeeExists method to check if the employee is already in the list



```
class EmployeeLinkedList {  
  
    private Employee head;  
  
    // Method to check if an employee exists in the linked list before insertion  
    public boolean employeeExists(String empId) {  
        Employee current = head; // Start from the head of the list  
        while (current != null) { // Traverse the list until the end  
            if (current.getEmpId().equals(empId)) { // If the employee ID matches, return true  
                return true;  
            }  
            current = current.getNext(); // Move to the next node in the list  
        }  
        return false; // If no match is found after traversing the list, return false  
    }  
}
```

### 2-insertEmployee method to add employee in sorted way

```
public void insertEmployee(Employee newEmployee) {  
    // Check if the employee already exists in the list  
    if (employeeExists(newEmployee.getEmpId())) {  
        System.out.println("Error: Employee with ID already exists."); // If so, print an error message and return  
        return;  
    }  
    // If the list is empty or the new employee's ID is less than the head's ID, insert at the beginning  
    if (head == null || newEmployee.getEmpId().compareTo(head.getEmpId()) < 0) {  
        newEmployee.setNext(head); // The new employee's next node is the current head  
        head = newEmployee; // The new employee becomes the head  
        return;  
    }  
    // If the new employee's ID is greater than the head's ID, find the correct position to insert  
    Employee current = head; // Start from the head of the list  
    // Traverse the list until the end or until finding a node with a greater ID  
    while (current.getNext() != null && current.getNext().getEmpId().compareTo(newEmployee.getEmpId()) < 0) {  
        current = current.getNext(); // Move to the next node in the list  
    }  
    // Insert the new employee after the current node  
    newEmployee.setNext(current.getNext()); // The new employee's next node is the current node's next node  
    current.setNext(newEmployee); // The current node's next node is the new employee  
}
```

### 3-deleteEmployee method to delete an employee based on their ID

```
// Method to delete an employee record by ID and return an integer
public int deleteEmployee(String empId) {
    // If the list is empty, return -1
    if (head == null) {
        return -1;
    }
    // If the head's employee ID matches the given ID, remove the head and return 0
    if (head.getEmpId().equals(anObject:empId)) {
        head = head.getNext();
        return 0;
    }
    // Start from the head of the list
    Employee current = head;
    // Traverse the list until the end
    while (current.getNext() != null) {
        // If the next node's employee ID matches the given ID, remove the next node and return 0
        if (current.getNext().getEmpId().equals(anObject:empId)) {
            current.setNext(current.getNext().getNext());
            return 0;
        }
        // Move to the next node in the list
        current = current.getNext();
    }
    // If no match is found after traversing the list, return -1
    return -1;
}
```

### 4- updateSalary method to update an employee salary based on worked hours

```
// Method to update the salary based on extra hours
public void updateSalary(String empId) {
    Employee current = head; // Start from the head of the list
    boolean found = false; // Flag to check if the employee is found
    // Traverse the list until the end
    while (current != null) {
        // If the employee ID matches, start the salary update process
        if (current.getEmpId().equals(anObject:empId)) {
            // Calculate the extra hours worked beyond 32 hours
            int extraHours = current.getWorkHours() - 32;
            // If there are extra hours, update the salary
            if (extraHours > 0) {
                // Calculate the increase in salary (2% of the current salary for each extra hour)
                double increase = current.getSalary() * 0.02 * extraHours;
                // Update the salary
                current.setSalary(current.getSalary() + increase);
                // Print the new salary
                //System.out.println("New Salary: " + current.getSalary());
                JOptionPane.showMessageDialog(parentComponent:null, "New Salary: " + current.getSalary());
            } else {
                // If there are no extra hours, print a message
                //System.out.println("No extra hours worked beyond 32 hours.");
                JOptionPane.showMessageDialog(parentComponent:null, message:"No extra hours worked beyond 32 hours.");
            }
            // Set the flag to true indicating the employee is found
            found = true;
            // Break the loop as the employee is found
            break;
        }
        // Move to the next node in the list
        current = current.getNext();
    }
    // If the employee is not found after traversing the list, print a message
    if (!found) {
        //System.out.println("Employee not found.");
        JOptionPane.showMessageDialog(parentComponent:null, message:"Employee not found.");
    }
}
```

## 5- method getHead to retrieve the head of the list

```
public Employee getHead() {  
    return head;  
}
```

## EmployeeForm class (UI)

1-btnAdd action listener to either add using the method in the linked list class or update information about an employee based on the text displayed in the button

```
btnAdd.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        if (validateFields()) {  
            String name = tfName.getText();  
            String id = tfID.getText();  
            String firstDay = tfFirstDay.getText();  
            String phone = tfPhone.getText();  
            String address = tfAddress.getText();  
            double salary = Double.parseDouble(tfSalary.getText());  
            int workHours = Integer.parseInt(tfWorkHours.getText());  
            if (btnAdd.getText().equalsIgnoreCase("add")) {  
                if (empList.employeeExists(empId: id)) {  
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Error: Employee with ID already exists.");  
                } else {  
                    Employee emp = new Employee(name, empId: id, firstDayOfWork: firstDay, phoneNumber: phone, address, workHours, salary);  
                    empList.insertEmployee(newEmployee: emp);  
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Successfully added!");  
                    Main main = new Main(list: empList);  
                    main.setVisible(b: true);  
                    dispose();  
                }  
            } else {  
                Employee current = empList.getHead();  
                while (current != null) {  
                    if (current.getEmpId().equalsIgnoreCase(employee.getEmpId())) {  
                        employee.setName(name);  
                        employee.setFirstDayOfWork(firstDayOfWork: firstDay);  
                        employee.setPhoneNumber(phoneNumber: phone);  
                        employee.setAddress(address);  
                        employee.setSalary(salary);  
                        employee.setWorkHours(workHours);  
                        JOptionPane.showMessageDialog(parentComponent: null, message: "Successfully Updated!");  
                        Main main = new Main(list: empList);  
                        main.setVisible(b: true);  
                        dispose();  
                        break;  
                    } else {  
                        current = current.getNext();  
                    }  
                }  
            }  
        }  
    }  
});
```

## 2- check if the employee parameter is not null to display the update interface

```
if (employee != null) {
    titleLabel.setText( text: "Update Employee");
    btnAdd.setText( text: "Update");
    tfID.setEnabled( enabled: false);
    tfID.setText( t: employee.getEmpId());
    tfName.setText( t: employee.getName());
    tfPhone.setText( t: employee.getPhoneNumber());
    tfAddress.setText( t: employee.getAddress());
    tfFirstDay.setText( t: employee.getFirstDayOfWork());
    tfWorkHours.setText( t: String.valueOf( i: employee.getWorkHours()));
    tfSalary.setText( t: String.valueOf( d: employee.getSalary()));
}
```

## 3-validation of the text fields

```
private boolean validateFields() {
    if (tfName.getText().isEmpty() || tfID.getText().isEmpty() || tfPhone.getText().isEmpty() || tfAddress.getText().isEmpty() || tfFirstDay.getText().isEmpty()) {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Please enter all fields!");
        return false;
    }

    try {
        Integer.parseInt( s: tfWorkHours.getText());
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Please enter valid data in Work Hours!");
        return false;
    }

    if (!isDouble( str: tfSalary.getText())) {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Please enter valid data in salary");
        return false;
    }

    if (!isValidPhoneNumber( phone: tfPhone.getText())) {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Please enter a valid phone number!");
        return false;
    }

    // Validate name
    if (!tfName.getText().matches( regex: "[a-zA-Z]+")) {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Please enter a valid name !");
        return false;
    }

    // Validate employee ID
    if (!tfID.getText().matches( regex: "\\d{7}")) {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Please enter a valid employee ID (exactly 7 digits)!");
        return false;
    }

    SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "dd/MM/yyyy"); //validate date
    dateFormat.setLenient( lenient: false);

    try {
        Date date = dateFormat.parse( source: tfFirstDay.getText());
    } catch (ParseException e) {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Please enter a valid date for the first day (dd/MM/yyyy)!");
        return false;
    }
}
```

## DetailsForm class (UI)

Used to display one or all the employee information on a table based on the value sent to the constructure.

```
History
public DetailsForm(EmployeeLinkedList empList, Employee employee) {
    setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);
    setBounds(x:100, y:100, width:763, height:529);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(top:5, left:5, bottom:5, right:5));
    setContentPane(contentPane);
    contentPane.setLayout(new BorderLayout(hgap:0, vgap:0));

    JScrollPane scrollPane = new JScrollPane();
    contentPane.add(comp: scrollPane, constraints: BorderLayout.CENTER);

    table = new JTable();
    scrollPane.setViewportView(view: table);

    if (employee != null) {
        String[] columnNames = {"Field", "Value"};
        String[][] rowData = {
            {"Name", employee.getName()},
            {"Employee ID", String.valueOf(obj.employee.getEmpId())},
            {"First Day of Work", employee.getFirstDayOfWork()},
            {"Phone Number", employee.getPhoneNumber()},
            {"Address", employee.getAddress()},
            {"Work Hours", String.valueOf(obj.employee.getWorkHours())},
            {"Salary", String.valueOf(obj.employee.getSalary()) + " SAR"}
        };
        DefaultTableModel model = new DefaultTableModel(data: rowData, columnNames);
        table.setModel(dataModel: model);
    } else {
        String[] columnNames = {"Employee ID", "Name", "First Day of Work", "Phone Number", "Address", "Work Hours", "Salary"};
        DefaultTableModel model = new DefaultTableModel(columnNames, rowCount: 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        };
        table.setModel(dataModel: model);

        Employee current = empList.getHead();
        while (current != null) {
            Object[] rowData = {
                current.getEmpId(),
                current.getName(),
                current.getFirstDayOfWork(),
                current.getPhoneNumber(),
                current.getAddress(),
                current.getWorkHours(),
                current.getSalary()
            };
            model.addRow(rowData);
            current = current.getNext();
        }

        JButton btnBack = new JButton(text: "Back");
        btnBack.setFont(new Font(name: "Times New Roman", style: Font.BOLD, size: 16));
        btnBack.addActionListener(e -> dispose()); // Close the form when back button is clicked
        contentPane.add(comp: btnBack, constraints: BorderLayout.SOUTH);

        btnBack.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                Main main = new Main(list: empList);
                main.setVisible(b: true);
                dispose();
            }
        });
    }
}
```



## Main Class (UI)

1-btnAdd to first display employee form gui and send the values to add employee

```
btnAddNew.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        EmployeeForm empForm = new EmployeeForm(empList:list, employee:null);  
        empForm.setVisible(b:true);  
        dispose();  
    }  
});
```

2-btnUpdate first ask the user to input employee id and see if the id is in the list, if its in the list then display the employee form gui and send the value to update

```
// method to update specific employee info  
btnUpdate.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        String id = JOptionPane.showInputDialog(parentComponent:null, message:"Please enter employee ID");  
        if (!id == null || id.isEmpty()) {  
            Employee current = list.getHead();  
            Employee emp = null;  
            while (current != null) {  
                if (current.getEmpId().equals(anObject:id)) {  
                    emp = current;  
                    break;  
                } else {  
                    current = current.getNext();  
                }  
            }  
            if (emp == null) {  
                JOptionPane.showMessageDialog(parentComponent:null, message:"Employee not found");  
            } else {  
                EmployeeForm empForm = new EmployeeForm(empList:list, employee:emp);  
                empForm.setVisible(b:true);  
                dispose();  
            }  
        }  
    }  
});
```

3-btnDelete use the method delete employee to delete an employee based on their id

```
btnDelete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String id = JOptionPane.showInputDialog(parentComponent:null, message: "Please enter employee ID");
        if (!(id == null || id.isEmpty())) {
            if (list.deleteEmployee(empId:id) == -1) {
                JOptionPane.showMessageDialog(parentComponent:null, message: "Employee not found!");
            } else {
                JOptionPane.showMessageDialog(parentComponent:null, message: "Successfully deleted!");
                saveToFile(emp: list.getHead());
            }
        }
    }
});
```

4-btnSearch to search for an employee based on their id

```
// this method show specific Employee Details
btnSearch.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        String id = JOptionPane.showInputDialog(parentComponent:null, message: "Please enter employee ID");
        if (!(id == null || id.isEmpty())) {
            Employee current = list.getHead();
            Employee emp = null;
            while (current != null) {
                if (current.getEmpId().equals(anObject:id)) {
                    emp = current;
                    break;
                } else {
                    current = current.getNext();
                }
            }
            if (emp == null) {
                JOptionPane.showMessageDialog(parentComponent:null, message: "Employee not found!");
            } else {
                DetailsForm form = new DetailsForm(empList:list, employee:emp);
                form.setVisible(b:true);
                dispose();
            }
        }
    }
});
```

## 5-btnShowAll to display all employee's information in a table

```
// this method show all Employee Details
btnShowAll.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (list.getHead() == null) {
            JOptionPane.showMessageDialog(parentComponent: null, message: "No employees found.");
        } else {
            DetailsForm form = new DetailsForm(empList: list, employee: null);
            form.setVisible(b: true);
            dispose();
        }
    }
});
```

## 6-btnUpdateSalary use the method update salary from the linked list class

```
btnUpdateSalary.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String id = JOptionPane.showInputDialog(parentComponent: null, "Enter employee ID and total hours worked to update salary:\n"
            + "Note: Salary will be updated based on extra hours worked beyond 32 hours.");
        if (!(id == null || id.isEmpty())) {
            // Call the updateSalary method passing the employee ID
            list.updateSalary(empId: id);
            // Save the updated list to file
            saveToFile(emp: list.getHead());
        }
    }
});
```