```
import java.util.*;
interface Stack<T> {
    void push(T item);
    T pop();
    boolean isEmpty();
}
class ArrayStack<T> implements Stack<T> {
    private Object[] stack;
    private int top;
    private int capacity;

    public ArrayStack(int capacity) {
        this.capacity = capacity;
        stack = new Object[capacity];
        top = -1;
    }

    public void push(T item) {
        if (top == capacity - 1) {
            System.out.println("Stack Overflow! Cannot save more states.");
            return;
        }
        stack[++top] = item;
    }

    @SuppressWarnings("unchecked")
    public T pop() {
        if (isEmpty()) return null;
        return (T) stack[top--];
    }

    public boolean isEmpty() {
        return top == -1;
    }
}

class Employee {
    int id;
    String name;
    double salary;

    Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    public String toString() {
```

```
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}
class PayrollSystem {
    private ArrayList<Employee> employees;
    private Stack<ArrayList<Employee>> undoStack;

    public PayrollSystem() {
        employees = new ArrayList<>();
        undoStack = new ArrayStack<>(50); // Stack to save previous states
    }
    private void saveState() {
        ArrayList<Employee> snapshot = new ArrayList<>(employees);
        undoStack.push(snapshot);
    }
    public void addEmployee(int id, String name, double salary) {
        try {
            if (salary < 0) throw new IllegalArgumentException("Salary cannot be negative.");
            saveState();
            employees.add(new Employee(id, name, salary));
            System.out.println("Employee added successfully.");
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
    public void calculateSalary() {
        try {
            if (employees.isEmpty()) throw new Exception("No employees to calculate salary.");
            double total = 0;
            for (Employee e : employees) {
                total += e.salary;
            }
            System.out.println("Total Salary to be paid: " + total);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
    public void undo() {
        if (!undoStack.isEmpty()) {
            employees = undoStack.pop();
            System.out.println("Undo successful! Last operation reverted.");
        } else {
            System.out.println("No operations to undo.");
        }
    }

public void displayEmployees() {
        if (employees.isEmpty()) {
```

```
            System.out.println("No employees found.");
        } else {
            System.out.println("\n--- Employee List ---");
            for (Employee e : employees) {
                System.out.println(e);
            }
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        PayrollSystem payroll = new PayrollSystem();
        int choice;

        do {
            System.out.println("\n--- Payroll System ---");
            System.out.println("1. Add Employee");
            System.out.println("2. Calculate Total Salary");
            System.out.println("3. Undo Last Operation");
            System.out.println("4. Display Employees");
            System.out.println("5. Exit");
            System.out.print("Enter choice: ");
            choice = sc.nextInt();
            sc.nextLine(); // consume newline

            switch (choice) {
                case 1:
                    try {
                        System.out.print("Enter Employee ID: ");
                        int id = sc.nextInt();
                        sc.nextLine();
                        System.out.print("Enter Employee Name: ");
                        String name = sc.nextLine();
                        System.out.print("Enter Salary: ");
                        double salary = sc.nextDouble();
                        payroll.addEmployee(id, name, salary);
                    } catch (InputMismatchException e) {
                        System.out.println("Invalid input. Please try again.");
                        sc.nextLine(); // clear invalid input
                    }
                    break;

                case 2:
                    payroll.calculateSalary();
                    break;

                case 3:
```

```
                payroll.undo();
                break;

            case 4:
                payroll.displayEmployees();
                break;

            case 5:
                System.out.println("Exiting Payroll System...");
                break;

            default:
                System.out.println("Invalid choice! Try again.");
        }
    } while (choice != 5);

    sc.close();
  }
}
```