```
import java.util.*;
import java.util.concurrent.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

// Employee class to store employee details
class Employee {
    private int id;
    private String name;
    private String department;
    private double salary;
    private String position;
    private LocalDateTime joinDate;

    public Employee(int id, String name, String department, double salary, String position) {
        this.id = id;
        this.name = name;
        this.department = department;
        this.salary = salary;
        this.position = position;
        this.joinDate = LocalDateTime.now();
    }

    // Getters
    public int getId() { return id; }
    public String getName() { return name; }
    public String getDepartment() { return department; }
    public double getSalary() { return salary; }
    public String getPosition() { return position; }
    public LocalDateTime getJoinDate() { return joinDate; }

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

    @Override
    public String toString() {
        return String.format("Employee[ID: %d, Name: %s, Dept: %s, Position: %s, Salary: $%.2f,
Joined: %s]",
                             id, name, department, position, salary,
                             joinDate.format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")));
    }

    public String toCSV() {
        return String.format("%d,%s,%s,%.2f,%s,%s",
                             id, name, department, salary, position, joinDate);
    }
}

// Thread class implementation (extends Thread)
class EmployeeProcessorThread extends Thread {
    private Employee employee;
    private String processType;

    public EmployeeProcessorThread(Employee employee, String processType) {
        this.employee = employee;
        this.processType = processType;
    }

    @Override
    public void run() {
        System.out.printf("[Thread-%d] ðŸš€ Starting %s processing for: %s%n",
```

```
                        Thread.currentThread().getId(), processType, employee.getName());

        try {
            // Simulate some processing time
            Thread.sleep(ThreadLocalRandom.current().nextInt(500, 2000));

            switch (processType) {
                case "SALARY_CALCULATION":
                    processSalary();
                    break;
                case "TAX_CALCULATION":
                    processTax();
                    break;
                case "BENEFITS_PROCESSING":
                    processBenefits();
                    break;
                case "PERFORMANCE_REVIEW":
                    processPerformance();
                    break;
                default:
                    System.out.printf("[Thread-%d] âš ï¸ Unknown process type: %s%n",
                                      Thread.currentThread().getId(), processType);
            }

            System.out.printf("[Thread-%d] âœ… Completed %s for: %s%n",
                              Thread.currentThread().getId(), processType, employee.getName());

        } catch (InterruptedException e) {
            System.out.printf("[Thread-%d] â❌ Processing interrupted for: %s%n",
                              Thread.currentThread().getId(), employee.getName());
            Thread.currentThread().interrupt();
        }
    }

    private void processSalary() {
        double bonus = employee.getSalary() * 0.1; // 10% bonus
        double newSalary = employee.getSalary() + bonus;
        System.out.printf("[Thread-%d] ðŸ’° Salary processed for %s: $%.2f -> $%.2f (Bonus: $
%.2f)%n",
                          Thread.currentThread().getId(), employee.getName(),
                          employee.getSalary(), newSalary, bonus);
    }

    private void processTax() {
        double tax = employee.getSalary() * 0.15; // 15% tax
        System.out.printf("[Thread-%d] ðŸ“Š Tax calculated for %s: $%.2f (15%% of $%.2f)%n",
                          Thread.currentThread().getId(), employee.getName(),
                          tax, employee.getSalary());
    }

    private void processBenefits() {
        System.out.printf("[Thread-%d] ðŸ¥ Benefits processed for %s (Dept: %s)%n",
                          Thread.currentThread().getId(), employee.getName(),
employee.getDepartment());
    }

    private void processPerformance() {
        String[] ratings = {"Excellent", "Good", "Average", "Needs Improvement"};
        String rating = ratings[ThreadLocalRandom.current().nextInt(ratings.length)];
        System.out.printf("[Thread-%d] ðŸ“ˆ Performance review for %s: %s%n",
                          Thread.currentThread().getId(), employee.getName(), rating);
    }
```

```
}

// Runnable interface implementation
class EmployeeProcessorRunnable implements Runnable {
    private Employee employee;
    private String processType;
    private CountDownLatch latch;

    public EmployeeProcessorRunnable(Employee employee, String processType, CountDownLatch
latch) {
        this.employee = employee;
        this.processType = processType;
        this.latch = latch;
    }

    @Override
    public void run() {
        System.out.printf("[Runnable-%d] ðŸš€ Starting %s for: %s%n",
                          Thread.currentThread().getId(), processType, employee.getName());

        try {
            // Simulate processing time
            Thread.sleep(ThreadLocalRandom.current().nextInt(300, 1500));

            // Perform different operations based on process type
            performOperation();

            System.out.printf("[Runnable-%d] âœ… Completed %s for: %s%n",
                              Thread.currentThread().getId(), processType, employee.getName());

        } catch (InterruptedException e) {
            System.out.printf("[Runnable-%d] â Processing interrupted for: %s%n",
                              Thread.currentThread().getId(), employee.getName());
            Thread.currentThread().interrupt();
        } finally {
            // Count down the latch to indicate completion
            if (latch != null) {
                latch.countDown();
            }
        }
    }

    private void performOperation() {
        switch (processType) {
            case "DATA_VALIDATION":
                validateEmployeeData();
                break;
            case "REPORT_GENERATION":
                generateReport();
                break;
            case "EMAIL_NOTIFICATION":
                sendNotification();
                break;
            case "BACKUP_PROCESSING":
                processBackup();
                break;
            default:
                System.out.printf("[Runnable-%d] âš ï¸ Unknown operation: %s%n",
                                  Thread.currentThread().getId(), processType);
        }
    }
```

```
    private void validateEmployeeData() {
        boolean isValid = employee.getId() > 0 &&
                          employee.getName() != null &&
                          !employee.getName().isEmpty() &&
                          employee.getSalary() > 0;

        System.out.printf("[Runnable-%d] ðŸ" Validation for %s: %s%n",
                          Thread.currentThread().getId(), employee.getName(),
                          isValid ? "PASSED â…" : "FAILED â");
    }

    private void generateReport() {
        System.out.printf("[Runnable-%d] ðŸ"‹ Report generated for %s (%s - %s)%n",
                          Thread.currentThread().getId(), employee.getName(),
                          employee.getDepartment(), employee.getPosition());
    }

    private void sendNotification() {
        System.out.printf("[Runnable-%d] ðŸ"§ Email sent to %s regarding %s position%n",
                          Thread.currentThread().getId(), employee.getName(),
employee.getPosition());
    }

    private void processBackup() {
        System.out.printf("[Runnable-%d] ðŸ'¾ Backup processed for employee: %s (ID: %d)%n",
                          Thread.currentThread().getId(), employee.getName(), employee.getId());
    }
}

// Employee Database to manage employees
class EmployeeDatabase {
    private List<Employee> employees;
    private int nextId;

    public EmployeeDatabase() {
        this.employees = new ArrayList<>();
        this.nextId = 1;
    }

    public void addEmployee(String name, String department, double salary, String position) {
        Employee employee = new Employee(nextId++, name, department, salary, position);
        employees.add(employee);
        System.out.println("â… Added: " + employee.getName());
    }

    public void addEmployee(Employee employee) {
        employees.add(employee);
        System.out.println("â… Added: " + employee.getName());
    }

    public List<Employee> getAllEmployees() {
        return new ArrayList<>(employees);
    }

    public Employee getEmployeeById(int id) {
        return employees.stream()
                        .filter(emp -> emp.getId() == id)
                        .findFirst()
                        .orElse(null);
    }

    public List<Employee> getEmployeesByDepartment(String department) {
```

```
        return employees.stream()
                        .filter(emp -> emp.getDepartment().equalsIgnoreCase(department))
                        .toList();
    }

    public int getEmployeeCount() {
        return employees.size();
    }

    public void displayAllEmployees() {
        System.out.println("\n=== ALL EMPLOYEES ===");
        if (employees.isEmpty()) {
            System.out.println("No employees in database.");
            return;
        }

        for (Employee emp : employees) {
            System.out.println(emp);
        }
    }
}

// Main class to demonstrate threading
public class MultiThreadedEmployeeSystem {
    private EmployeeDatabase database;
    private Scanner scanner;

    public MultiThreadedEmployeeSystem() {
        this.database = new EmployeeDatabase();
        this.scanner = new Scanner(System.in);
        initializeSampleData();
    }

    private void initializeSampleData() {
        // Add some sample employees
        database.addEmployee("John Smith", "Engineering", 75000, "Software Engineer");
        database.addEmployee("Jane Doe", "Marketing", 65000, "Marketing Manager");
        database.addEmployee("Bob Johnson", "HR", 60000, "HR Specialist");
        database.addEmployee("Alice Brown", "Engineering", 80000, "Senior Developer");
        database.addEmployee("Charlie Wilson", "Sales", 55000, "Sales Representative");
        database.addEmployee("Diana Lee", "Finance", 70000, "Financial Analyst");
        database.addEmployee("Mike Davis", "Engineering", 90000, "Tech Lead");
        database.addEmployee("Sarah Miller", "Marketing", 62000, "Content Writer");
    }

    public void start() {
        System.out.println("ðŸš€ Multi-threaded Employee Management System Started!");

        boolean running = true;
        while (running) {
            displayMenu();
            int choice = getIntInput("Choose an option: ");

            switch (choice) {
                case 1:
                    addNewEmployee();
                    break;
                case 2:
                    displayAllEmployees();
                    break;
                case 3:
                    processWithThreadClass();
```

```
                    break;
                case 4:
                    processWithRunnableInterface();
                    break;
                case 5:
                    processWithThreadPool();
                    break;
                case 6:
                    compareThreadingMethods();
                    break;
                case 7:
                    running = false;
                    System.out.println("ðŸ‘‹ Thank you for using the Employee Management
System!");
                    break;
                default:
                    System.out.println("â Invalid option! Please try again.");
            }
        }

        scanner.close();
    }

    private void displayMenu() {
        System.out.println("\n=== EMPLOYEE MANAGEMENT SYSTEM ===");
        System.out.println("1. Add New Employee");
        System.out.println("2. Display All Employees");
        System.out.println("3. Process Employees (Thread Class)");
        System.out.println("4. Process Employees (Runnable Interface)");
        System.out.println("5. Process Employees (Thread Pool)");
        System.out.println("6. Compare Threading Methods");
        System.out.println("7. Exit");
    }

    private void addNewEmployee() {
        System.out.println("\n--- Add New Employee ---");

        System.out.print("Enter name: ");
        String name = scanner.nextLine();

        System.out.print("Enter department: ");
        String department = scanner.nextLine();

        System.out.print("Enter salary: ");
        double salary = getDoubleInput();

        System.out.print("Enter position: ");
        String position = scanner.nextLine();

        database.addEmployee(name, department, salary, position);
    }

    private void displayAllEmployees() {
        database.displayAllEmployees();
    }

    private void processWithThreadClass() {
        System.out.println("\n--- Processing with Thread Class ---");
        List<Employee> employees = database.getAllEmployees();

        String[] processTypes = {"SALARY_CALCULATION", "TAX_CALCULATION", "BENEFITS_PROCESSING",
"PERFORMANCE_REVIEW"};
```

```
        System.out.printf("Starting %d threads using Thread class...%n", employees.size());

        List<Thread> threads = new ArrayList<>();

        for (int i = 0; i < employees.size(); i++) {
            Employee employee = employees.get(i);
            String processType = processTypes[i % processTypes.length];

            EmployeeProcessorThread processor = new EmployeeProcessorThread(employee,
processType);
            threads.add(processor);
            processor.start();
        }

        // Wait for all threads to complete
        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                System.out.println("â Main thread interrupted while waiting for worker
threads");
                Thread.currentThread().interrupt();
                return;
            }
        }

        System.out.println("ðŸŽ‰ All Thread class processing completed!");
    }

    private void processWithRunnableInterface() {
        System.out.println("\n--- Processing with Runnable Interface ---");
        List<Employee> employees = database.getAllEmployees();

        String[] processTypes = {"DATA_VALIDATION", "REPORT_GENERATION", "EMAIL_NOTIFICATION",
"BACKUP_PROCESSING"};

        System.out.printf("Starting %d threads using Runnable interface...%n",
employees.size());

        CountDownLatch latch = new CountDownLatch(employees.size());
        List<Thread> threads = new ArrayList<>();

        for (int i = 0; i < employees.size(); i++) {
            Employee employee = employees.get(i);
            String processType = processTypes[i % processTypes.length];

            EmployeeProcessorRunnable runnable = new EmployeeProcessorRunnable(employee,
processType, latch);
            Thread thread = new Thread(runnable);
            threads.add(thread);
            thread.start();
        }

        // Wait for all threads to complete using CountDownLatch
        try {
            latch.await();
            System.out.println("ðŸŽ‰ All Runnable interface processing completed!");
        } catch (InterruptedException e) {
            System.out.println("â Main thread interrupted while waiting for worker threads");
            Thread.currentThread().interrupt();
        }
```

```
    }

    private void processWithThreadPool() {
        System.out.println("\n--- Processing with Thread Pool Executor ---");
        List<Employee> employees = database.getAllEmployees();

        // Create a thread pool with fixed number of threads
        int poolSize = Math.min(4, employees.size()); // Maximum 4 threads
        ExecutorService executor = Executors.newFixedThreadPool(poolSize);

        System.out.printf("Using thread pool with %d threads for %d employees%n", poolSize,
employees.size());

        String[] processTypes = {"SALARY_CALCULATION", "REPORT_GENERATION", "DATA_VALIDATION",
"PERFORMANCE_REVIEW"};

        List<Future<?>> futures = new ArrayList<>();

        for (int i = 0; i < employees.size(); i++) {
            Employee employee = employees.get(i);
            String processType = processTypes[i % processTypes.length];

            // For thread pool, we use Runnable implementation
            EmployeeProcessorRunnable task = new EmployeeProcessorRunnable(employee,
processType, null);
            Future<?> future = executor.submit(task);
            futures.add(future);
        }

        // Shutdown the executor and wait for completion
        executor.shutdown();

        try {
            if (executor.awaitTermination(1, TimeUnit.MINUTES)) {
                System.out.println("ðŸŽ‰ Thread pool processing completed successfully!");
            } else {
                System.out.println("âš ï¸ Thread pool processing timed out!");
            }
        } catch (InterruptedException e) {
            System.out.println("âŒ Thread pool processing interrupted");
            Thread.currentThread().interrupt();
        }
    }

    private void compareThreadingMethods() {
        System.out.println("\n--- Comparing Threading Methods ---");

        List<Employee> employees = database.getAllEmployees().subList(0, 3); // Use first 3
employees for comparison

        System.out.println("Comparing performance for " + employees.size() + " employees:");

        // Test Thread class
        long startTime = System.currentTimeMillis();
        testThreadClass(employees);
        long threadTime = System.currentTimeMillis() - startTime;

        // Test Runnable interface
        startTime = System.currentTimeMillis();
        testRunnableInterface(employees);
        long runnableTime = System.currentTimeMillis() - startTime;
```

```
        // Test Thread Pool
        startTime = System.currentTimeMillis();
        testThreadPool(employees);
        long poolTime = System.currentTimeMillis() - startTime;

        System.out.println("\n=== PERFORMANCE COMPARISON ===");
        System.out.printf("Thread Class: %d ms%n", threadTime);
        System.out.printf("Runnable Interface: %d ms%n", runnableTime);
        System.out.printf("Thread Pool: %d ms%n", poolTime);

        String fastest = "Thread Pool";
        long minTime = Math.min(Math.min(threadTime, runnableTime), poolTime);
        if (minTime == threadTime) fastest = "Thread Class";
        else if (minTime == runnableTime) fastest = "Runnable Interface";

        System.out.printf("ðŸ† Fastest method: %s%n", fastest);
    }

    private void testThreadClass(List<Employee> employees) {
        List<Thread> threads = new ArrayList<>();
        for (Employee emp : employees) {
            EmployeeProcessorThread thread = new EmployeeProcessorThread(emp,
"PERFORMANCE_REVIEW");
            threads.add(thread);
            thread.start();
        }

        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }

    private void testRunnableInterface(List<Employee> employees) {
        CountDownLatch latch = new CountDownLatch(employees.size());
        List<Thread> threads = new ArrayList<>();

        for (Employee emp : employees) {
            EmployeeProcessorRunnable runnable = new EmployeeProcessorRunnable(emp,
"DATA_VALIDATION", latch);
            Thread thread = new Thread(runnable);
            threads.add(thread);
            thread.start();
        }

        try {
            latch.await();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    private void testThreadPool(List<Employee> employees) {
        ExecutorService executor = Executors.newFixedThreadPool(2);
        CountDownLatch latch = new CountDownLatch(employees.size());

        for (Employee emp : employees) {
            EmployeeProcessorRunnable task = new EmployeeProcessorRunnable(emp,
"REPORT_GENERATION", latch);
```

```
            executor.submit(task);
        }

        executor.shutdown();
        try {
            executor.awaitTermination(1, TimeUnit.MINUTES);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    // Utility methods for input handling
    private int getIntInput(String prompt) {
        while (true) {
            System.out.print(prompt);
            try {
                int input = scanner.nextInt();
                scanner.nextLine(); // Consume newline
                return input;
            } catch (InputMismatchException e) {
                System.out.println("â Please enter a valid number!");
                scanner.nextLine(); // Clear invalid input
            }
        }
    }

    private double getDoubleInput() {
        while (true) {
            try {
                double input = scanner.nextDouble();
                scanner.nextLine(); // Consume newline
                return input;
            } catch (InputMismatchException e) {
                System.out.println("â Please enter a valid number!");
                scanner.nextLine(); // Clear invalid input
            }
        }
    }

    public static void main(String[] args) {
        MultiThreadedEmployeeSystem system = new MultiThreadedEmployeeSystem();
        system.start();
    }
}
```