# Object Oriented Programming in Java Lab

## Manav Rachna International Institute of Research and Studies

### School of Computer Applications

| Submitted By | |
|---|---|
| Student Name | Rahul Sehraya |
| Roll No | 25/SCA/MCAN/081 |
| Programme | Masters of Computer Applications |
| Semester | 1th Semester |
| Section | B |
| Department | School of Computer Applications |
| Batch | 2025-27 |
| | |
| **Submitted To** | |
| Faculty Name | Dr. Ritu Sachdeva |

MANAV RACHNA vidyanatariksha | NAAC A++ ACCREDITED

Manav Rachna Campus Rd, Gadakhor Basti Village, Sector 43, Faridabad, Haryana 121004

**Java Minor Project**
**Rahul Sehraya(25/SCA/MCAN/081)**

Que : **Personal Task Reminder Manager that allows users to create tasks, set reminders, and receive notifications for upcoming deadlines**

Code :
```
import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.Panel;
import java.awt.TextArea;
import java.awt.TextField;
import java.awt.Toolkit;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.time.Duration;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class MinorProject {
   public static void main(String[] args) {
      TaskManager taskManager = new TaskManager();
      EventQueue.invokeLater(() -> {
         TaskReminderFrame frame = new TaskReminderFrame(taskManager);
         frame.setVisible(true);
      });
   }
}

class TaskReminderFrame extends Frame {
   private static final DateTimeFormatter DATE_FORMAT =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
```

```
    private final TaskManager taskManager;
    private final ReminderEngine reminderEngine;
    private final TextField titleField = new TextField();
    private final TextArea descriptionArea = new TextArea("", 3, 40,
TextArea.SCROLLBARS_VERTICAL_ONLY);
    private final TextField dueField = new TextField();
    private final TextField leadField = new TextField("30");
    private final TextField taskIdField = new TextField();
    private final TextArea tasksArea = new TextArea("", 12, 80,
TextArea.SCROLLBARS_VERTICAL_ONLY);
    private final TextArea notificationsArea = new TextArea("", 6, 80,
TextArea.SCROLLBARS_VERTICAL_ONLY);
    private final Label statusLabel = new Label("Ready");

    TaskReminderFrame(TaskManager taskManager) {
        super("Personal Task Reminder Manager");
        this.taskManager = taskManager;
        this.reminderEngine = new ReminderEngine(taskManager, new AwtNotifier(this));
        setSize(900, 650);
        setLayout(new BorderLayout(10, 10));
        buildForm();
        buildTaskPanels();
        setInitialValues();
        reminderEngine.start();
        refreshTaskList();
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                shutdown();
            }
        });
    }

    private void buildForm() {
        Panel formPanel = new Panel(new GridLayout(0, 2, 6, 6));
        formPanel.add(new Label("Title"));
        formPanel.add(titleField);
        formPanel.add(new Label("Description"));
        formPanel.add(descriptionArea);
        formPanel.add(new Label("Due (yyyy-MM-dd HH:mm)"));
        formPanel.add(dueField);
        formPanel.add(new Label("Reminder lead (minutes)"));
        formPanel.add(leadField);
        formPanel.add(new Label("Task ID (for complete/remove)"));
        formPanel.add(taskIdField);
        Panel buttonPanel = new Panel(new FlowLayout(FlowLayout.LEFT, 8, 4));
        Button addButton = new Button("Create Task");
        Button refreshButton = new Button("Refresh");
        Button completeButton = new Button("Mark Complete");
        Button removeButton = new Button("Remove Task");
        Button upcomingButton = new Button("Show Next Hour");
        Button clearNotifButton = new Button("Clear Notifications");
        buttonPanel.add(addButton);
        buttonPanel.add(refreshButton);
        buttonPanel.add(completeButton);
        buttonPanel.add(removeButton);
        buttonPanel.add(upcomingButton);
        buttonPanel.add(clearNotifButton);
        add(formPanel, BorderLayout.NORTH);
        add(buttonPanel, BorderLayout.CENTER);
        addButton.addActionListener(e -> handleCreateTask());
        refreshButton.addActionListener(e -> refreshTaskList());
        completeButton.addActionListener(e -> handleCompletion(true));
        removeButton.addActionListener(e -> handleCompletion(false));
        upcomingButton.addActionListener(e -> showUpcomingReminders());
        clearNotifButton.addActionListener(e -> notificationsArea.setText(""));
    }

    private void buildTaskPanels() {
        tasksArea.setEditable(false);
        notificationsArea.setEditable(false);
        Panel outputPanel = new Panel(new GridLayout(0, 1, 8, 8));
        Panel taskPanel = new Panel(new BorderLayout());
        taskPanel.add(new Label("Tasks"), BorderLayout.NORTH);
        taskPanel.add(tasksArea, BorderLayout.CENTER);
        Panel notificationPanel = new Panel(new BorderLayout());
        notificationPanel.add(new Label("Notifications"), BorderLayout.NORTH);
        notificationPanel.add(notificationsArea, BorderLayout.CENTER);
        outputPanel.add(taskPanel);
        outputPanel.add(notificationPanel);
        add(outputPanel, BorderLayout.SOUTH);
        add(statusLabel, BorderLayout.WEST);
    }

    private void setInitialValues() {
        LocalDateTime defaultDue = LocalDateTime.now().plusHours(1);
        dueField.setText(DATE_FORMAT.format(defaultDue));
        leadField.setText("30");
    }

    private void handleCreateTask() {
        String title = titleField.getText().trim();
        String description = descriptionArea.getText().trim();
        String dueInput = dueField.getText().trim();

        String leadInput = leadField.getText().trim();
        if (title.isEmpty()) {
            updateStatus("Title cannot be empty.");
            return;
        }
        try {
            LocalDateTime dueDate = LocalDateTime.parse(dueInput, DATE_FORMAT);
            if (dueDate.isBefore(LocalDateTime.now())) {
                updateStatus("Due date must be in the future.");
                return;
            }
            long leadMinutes = Long.parseLong(leadInput);
            if (leadMinutes < 0) {
                updateStatus("Reminder lead time cannot be negative.");
                return;
            }
            Task task = taskManager.addTask(title, description, dueDate,
Duration.ofMinutes(leadMinutes));
            updateStatus("Task created with ID: " + task.getId());
            clearForm();
            refreshTaskList();
        } catch (DateTimeParseException ex) {
            updateStatus("Invalid date format. Use yyyy-MM-dd HH:mm.");
        } catch (NumberFormatException ex) {
            updateStatus("Invalid number for reminder lead time.");
        }
    }

    private void handleCompletion(boolean markComplete) {
        String idText = taskIdField.getText().trim();
        if (idText.isEmpty()) {
            updateStatus("Enter task ID first.");
            return;
        }
        try {
            int id = Integer.parseInt(idText);
            if (markComplete) {
                Optional<Task> task = taskManager.markComplete(id);
                updateStatus(task.isPresent() ? "Task marked complete." : "Task ID not found.");
            } else {
                boolean removed = taskManager.removeTask(id);
                updateStatus(removed ? "Task removed." : "Task ID not found.");
            }

            refreshTaskList();
    } catch (NumberFormatException ex) {
        updateStatus("Invalid task ID.");
    }
}

private void showUpcomingReminders() {
    List<Task> upcoming = taskManager.getDueWithin(Duration.ofHours(1));
    if (upcoming.isEmpty()) {
        updateStatus("No reminders due in the next hour.");
        return;
    }
    StringBuilder sb = new StringBuilder("Reminders within 60 minutes:\n");
    for (Task task : upcoming) {
        sb.append(" - ").append(task.summaryLine()).append('\n');
    }
    notificationsArea.append(sb.toString());
    updateStatus("Listed reminders due within the hour.");
}

private void refreshTaskList() {
    List<Task> tasks = taskManager.getAllTasksSorted();
    if (tasks.isEmpty()) {
        tasksArea.setText("No tasks available.");
        return;
    }
    StringBuilder sb = new StringBuilder();
    for (Task task : tasks) {
        sb.append(task.describe()).append("\n\n");
    }
    tasksArea.setText(sb.toString());
}

void showNotification(String message) {
    String timestamp = LocalDateTime.now().format(DATE_FORMAT);
    notificationsArea.append("[" + timestamp + "] " + message + System.lineSeparator());
    Toolkit.getDefaultToolkit().beep();
}

private void updateStatus(String message) {
    statusLabel.setText(message);
}

private void clearForm() {
    titleField.setText("");
    descriptionArea.setText("");

    taskIdField.setText("");
    setInitialValues();
    }

    private void shutdown() {
        reminderEngine.stop();
        dispose();
        System.exit(0);
    }
}

class TaskManager {
    private final ConcurrentMap<Integer, Task> tasks = new ConcurrentHashMap<>();
    private int nextId = 1;

    public synchronized Task addTask(String title, String description, LocalDateTime
dueDateTime, Duration reminderLead) {
        int id = nextId++;
        Task task = new Task(id, title, description, dueDateTime, reminderLead);
        tasks.put(id, task);
        return task;
    }

    public List<Task> getAllTasksSorted() {
        List<Task> list = new ArrayList<>(tasks.values());
        list.sort(Comparator.comparing(Task::getDueDateTime));
        return list;
    }

    public Optional<Task> markComplete(int id) {
        Task task = tasks.get(id);
        if (task != null) {
            task.markComplete();
            return Optional.of(task);
        }
        return Optional.empty();
    }

    public boolean removeTask(int id) {
        return tasks.remove(id) != null;
    }

    public List<Task> getDueWithin(Duration window) {
        LocalDateTime now = LocalDateTime.now();
        LocalDateTime threshold = now.plus(window);
        List<Task> results = new ArrayList<>();

        for (Task task : tasks.values()) {
            if (!task.isCompleted() && !task.isPastDue(now) &&
!task.getReminderTime().isBefore(now) && !task.getReminderTime().isAfter(threshold)) {
```

```
                results.add(task);
            }
        }
        results.sort(Comparator.comparing(Task::getReminderTime));
        return results;
    }

    public List<Task> getActiveTasks() {
        return new ArrayList<>(tasks.values());
    }
}

class Task {
    private static final DateTimeFormatter OUTPUT_FORMAT =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
    private final int id;
    private final String title;
    private final String description;
    private final LocalDateTime dueDateTime;
    private final Duration reminderLeadTime;
    private volatile boolean completed;
    private volatile boolean reminderSent;
    private volatile boolean deadlineAlerted;

    Task(int id, String title, String description, LocalDateTime dueDateTime, Duration
reminderLeadTime) {
        this.id = id;
        this.title = title.isBlank() ? "Untitled Task" : title;
        this.description = description == null ? "" : description;
        this.dueDateTime = dueDateTime;
        this.reminderLeadTime = reminderLeadTime;
        this.completed = false;
        this.reminderSent = false;
        this.deadlineAlerted = false;
    }

    public int getId() {
        return id;
    }

    public LocalDateTime getDueDateTime() {
        return dueDateTime;
    }

    public LocalDateTime getReminderTime() {
        LocalDateTime reminderTime = dueDateTime.minus(reminderLeadTime);
        return reminderTime.isAfter(dueDateTime) ? dueDateTime : reminderTime;
    }
```

```
public boolean isCompleted() {
    return completed;
}

public boolean shouldTriggerReminder(LocalDateTime now) {
    return !completed && !reminderSent && !now.isBefore(getReminderTime());
}

public boolean shouldTriggerDeadlineAlert(LocalDateTime now) {
    return !completed && !deadlineAlerted && !now.isBefore(dueDateTime);
}

public boolean isPastDue(LocalDateTime now) {
    return now.isAfter(dueDateTime);
}

public void markReminderSent() {
    reminderSent = true;
}

public void markDeadlineAlerted() {
    deadlineAlerted = true;
}

public void markComplete() {
    completed = true;
}

public String describe() {
    StringBuilder sb = new StringBuilder();
    sb.append("[").append(id).append("] ").append(title);
    if (!description.isBlank()) {
        sb.append(" - ").append(description);
    }
    sb.append("\n   Due: ").append(OUTPUT_FORMAT.format(dueDateTime));
    sb.append(" | Reminder: ").append(OUTPUT_FORMAT.format(getReminderTime()));
    sb.append(" | Status: ").append(completed ? "Completed" : "Active");
    return sb.toString();
}

public String summaryLine() {

    return "[" + id + "] " + title + " (due " + OUTPUT_FORMAT.format(dueDateTime) + ")";
}

public String describeReminderWindow() {
    Duration untilReminder = Duration.between(LocalDateTime.now(), getReminderTime());
```

```
        long minutes = Math.max(untilReminder.toMinutes(), 0);
        return "[" + id + "] " + title + " -> reminder in " + minutes + " minutes (due " +
OUTPUT_FORMAT.format(dueDateTime) + ")";
    }
}

class ReminderEngine {
    private final TaskManager taskManager;
    private final Notifier notifier;
    private final ScheduledExecutorService scheduler =
Executors.newSingleThreadScheduledExecutor();
    private final long intervalSeconds;

    ReminderEngine(TaskManager taskManager, Notifier notifier) {
        this(taskManager, notifier, 30);
    }

    ReminderEngine(TaskManager taskManager, Notifier notifier, long intervalSeconds) {
        this.taskManager = taskManager;
        this.notifier = notifier;
        this.intervalSeconds = Math.max(5, intervalSeconds);
    }

    public void start() {
        scheduler.scheduleAtFixedRate(this::scanAndNotify, 5, intervalSeconds,
TimeUnit.SECONDS);
    }

    public void stop() {
        scheduler.shutdownNow();
    }

    private void scanAndNotify() {
        LocalDateTime now = LocalDateTime.now();
        for (Task task : taskManager.getActiveTasks()) {
            if (task.shouldTriggerReminder(now)) {
                notifier.notifyTask(task, "Reminder window reached");
                task.markReminderSent();
            }
            if (task.shouldTriggerDeadlineAlert(now)) {
                notifier.notifyTask(task, "Deadline reached");

                task.markDeadlineAlerted();
            }
        }
    }
}
```

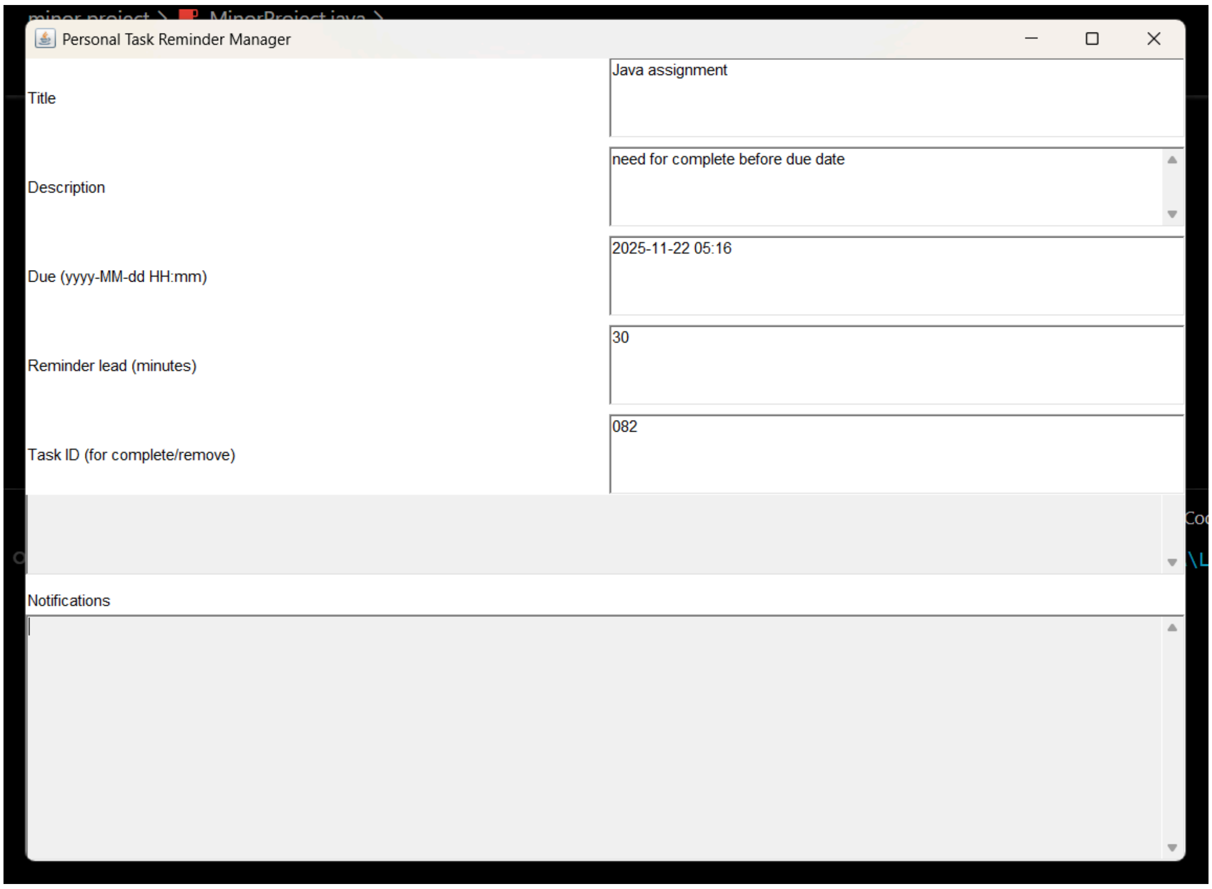
```
interface Notifier {
   void notifyTask(Task task, String message);
}

class AwtNotifier implements Notifier {
   private final TaskReminderFrame frame;

   AwtNotifier(TaskReminderFrame frame) {
      this.frame = frame;
   }

   @Override
   public void notifyTask(Task task, String message) {
      EventQueue.invokeLater(() -> frame.showNotification(message + " -> " +
task.summaryLine()));
   }
}
```

Output:


Personal Task Reminder Manager

| | |
|---|---|
| Title | Java assignment |
| Description | need for complete before due date |
| Due (yyyy-MM-dd HH:mm) | 2025-11-22 05:16 |
| Reminder lead (minutes) | 30 |
| Task ID (for complete/remove) | 082 |

Notifications