**OOP Project Progress**

**Assignment 3**

**Submitted by**

**Member 1**

Muhammad Ahad Ata

FA24-BCS-007

**Member 2**

Ahmer Ijaz

FA24-BCS-010

**Section:** A

**Submitted To:** Sir Shahid Bhatti

**Date:** 24th March 2025

**Smart Academic Planner**

**Abstract**

Smart Academic Planner is a Java-based system that helps students manage their assignments and plan their studies. Students can add assignments with subject, due date, and description, and update their status as "pending" or "completed." The system uses Gemini AI to generate a personalized study plan based on pending assignments and deadlines, suggesting what subjects to study and when. Additionally, the system tracks the history of completed assignments, allowing students to review their progress. It also provides Statistics and Reports on completed and pending assignments, helping students track time spent and improve their study habits.

We also make efforts to design a user-friendly UI/UX that enhances the overall experience, ensuring ease of use and seamless interaction with the system, though this feature remains optional

**Task Creation**

**Assignment Task**

package TaskCreation;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

public class Assignment\_task extends TaskCreation {

private String course\_name;

private Date submission\_date;

public Assignment\_task(String task\_name, Date due\_date, String priority, boolean isComplete, String course\_name, Date submission\_date) {

super(task\_name, due\_date, priority, isComplete);

this.course\_name = course\_name;

this.submission\_date = submission\_date;

setDue\_date(submission\_date);

}

public String getCourse\_name() {

return course\_name;

}

public void setCourse\_name(String course\_name) {

this.course\_name = course\_name;

}

public Date getSubmission\_date() {

return submission\_date;

}

public void setSubmission\_date(Date submission\_date) {

this.submission\_date = submission\_date;

setDue\_date(submission\_date);

}

@Override

public void setDue\_date(Date submission\_date) {

Calendar cal = Calendar.getInstance();

cal.setTime(submission\_date);

cal.add(Calendar.DATE, -1);

super.setDue\_date(cal.getTime());

}

@Override

public String toString() {

SimpleDateFormat formatter = new SimpleDateFormat("dd-MMM-yyyy hh:mm a");

return super.toString() + ", Course: " + course\_name + ", Submission Date: " + formatter.format(submission\_date);

}

}

**General Task**

package TaskCreation;

import java.text.SimpleDateFormat;

import java.util.Date;

public class General\_task extends TaskCreation {

private String task\_description;

public General\_task(String task\_name, Date due\_date, String priority, boolean isComplete, String task\_description) {

super(task\_name, due\_date, priority, isComplete);

this.task\_description = task\_description;

}

public String getTask\_description() {

return task\_description;

}

public void setTask\_description(String task\_description) {

this.task\_description = task\_description;

}

@Override

public String toString() {

SimpleDateFormat formatter = new SimpleDateFormat("dd-MMM-yyyy hh:mm a");

return super.toString() + ", Task Description: " + task\_description + ", Due Date: " + formatter.format(getDue\_date());

}

}

**Exam Task**

package TaskCreation;

import java.text.SimpleDateFormat;

import java.util.Date;

public class General\_task extends TaskCreation {

private String task\_description;

public General\_task(String task\_name, Date due\_date, String priority, boolean isComplete, String task\_description) {

super(task\_name, due\_date, priority, isComplete);

this.task\_description = task\_description;

}

public String getTask\_description() {

return task\_description;

}

public void setTask\_description(String task\_description) {

this.task\_description = task\_description;

}

@Override

public String toString() {

SimpleDateFormat formatter = new SimpleDateFormat("dd-MMM-yyyy hh:mm a");

return super.toString() + ", Task Description: " + task\_description + ", Due Date: " + formatter.format(getDue\_date());

}

}

**Task Creation**

package TaskCreation;

import java.text.SimpleDateFormat;

import java.util.Date;

public class TaskCreation {

private static int idCounter = 1;

private int taskId;

private String task\_name;

private Date due\_date;

private String priority;

private boolean isComplete;

public TaskCreation(String task\_name, Date due\_date, String priority, boolean isComplete) {

this.taskId = idCounter++;

this.task\_name = task\_name;

this.due\_date = due\_date;

this.priority = priority;

this.isComplete = isComplete;

}

public int getTaskId() {

return taskId;

}

public void setTaskId(int id) {

this.taskId = id;

}

public String getTask\_name() {

return task\_name;

}

public void setTask\_name(String task\_name) {

this.task\_name = task\_name;

}

public Date getDue\_date() {

return due\_date;

}

public void setDue\_date(Date due\_date) {

this.due\_date = due\_date;

}

public String getPriority() {

return priority;

}

public void setPriority(String priority) {

this.priority = priority;

}

public boolean isComplete() {

return isComplete;

}

public void setComplete(boolean complete) {

isComplete = complete;

}

public void markComplete() {

this.isComplete = true;

}

SimpleDateFormat formatter = new SimpleDateFormat("dd-MMM-yyyy hh:mm a");

@Override

public String toString() {

return "Task ID: " + taskId +

" | Task: " + task\_name +

" | Due: " + formatter.format(due\_date) +

" | Priority: " + priority +

" | Completed: " + isComplete;

}

}

**Task Creation Main**

package TaskCreation;

import MarkCompleteTask.MarkComplete;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import java.util.Scanner;

public class TaskCreationMain {

public static List<TaskCreation> tasklist = new ArrayList<>();

public static void displayMenu() {

System.out.println("Choose a task type to create:");

System.out.println("1. Exam Task");

System.out.println("2. Assignment Task");

System.out.println("3. General Task");

System.out.println("4. View All Tasks");

System.out.println("5. Mark Task Complete or Reopen");

}

public static void handleUserInput(int choice, List<TaskCreation> tasklist) {

switch (choice) {

case 1 -> createExamTask(tasklist);

case 2 -> createAssignmentTask(tasklist);

case 3 -> createGeneralTask(tasklist);

case 4 -> viewAllTasks(tasklist);

case 5 -> MarkComplete.markCompleteOrReopen(tasklist);

default -> System.out.println("Invalid choice, please try again.");

}

}

public static void createExamTask(List<TaskCreation> taskList) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter task name: ");

String taskName = scanner.nextLine();

System.out.print("Enter due date and time (dd-MM-yyyy hh:mm AM/PM): ");

String dueDateString = scanner.nextLine();

Date dueDate = parseDate(dueDateString);

System.out.print("Enter priority: ");

String priority = scanner.nextLine();

System.out.print("Is the task complete (true/false): ");

boolean isComplete = Boolean.parseBoolean(scanner.nextLine());

System.out.print("Enter subject name: ");

String subjectName = scanner.nextLine();

System.out.print("Enter exam date (dd-MM-yyyy): ");

String examDateString = scanner.nextLine();

Date examDate = parseDate(examDateString);

Exam\_task examTask = new Exam\_task(taskName, dueDate, priority, isComplete, subjectName, examDate);

tasklist.add(examTask);

System.out.println("Exam Task Created Successfully!");

}

public static void createAssignmentTask(List<TaskCreation> taskList) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter task name: ");

String taskName = scanner.nextLine();

System.out.print("Enter due date and time (dd-MM-yyyy hh:mm AM/PM): ");

String dueDateString = scanner.nextLine();

Date dueDate = parseDate(dueDateString);

System.out.print("Enter priority: ");

String priority = scanner.nextLine();

System.out.print("Is the task complete (true/false): ");

boolean isComplete = Boolean.parseBoolean(scanner.nextLine());

System.out.print("Enter course name: ");

String courseName = scanner.nextLine();

System.out.print("Enter submission date (dd-MM-yyyy): ");

String submissionDateString = scanner.nextLine();

Date submissionDate = parseDate(submissionDateString);

Assignment\_task assignmentTask = new Assignment\_task(taskName, dueDate, priority, isComplete, courseName, submissionDate);

tasklist.add(assignmentTask);

System.out.println("Assignment Task Created Successfully!");

}

public static void createGeneralTask(List<TaskCreation> taskList) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter task name: ");

String taskName = scanner.nextLine();

System.out.print("Enter due date and time (dd-MM-yyyy hh:mm AM/PM): ");

String dueDateString = scanner.nextLine();

Date dueDate = parseDate(dueDateString);

System.out.print("Enter priority: ");

String priority = scanner.nextLine();

System.out.print("Is the task complete (true/false): ");

boolean isComplete = Boolean.parseBoolean(scanner.nextLine());

System.out.print("Enter task description: ");

String taskDescription = scanner.nextLine();

General\_task generalTask = new General\_task(taskName, dueDate, priority, isComplete, taskDescription);

tasklist.add(generalTask);

System.out.println("General Task Created Successfully!");

}

public static Date parseDate(String dateString) {

try {

SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy hh:mm a");

return formatter.parse(dateString);

} catch (Exception e) {

System.out.println("Invalid date format. Using today's date instead.");

return new Date();

}

}

public static void viewAllTasks(List<TaskCreation> taskList) {

if (tasklist.isEmpty()) {

System.out.println("No tasks available.");

} else {

for (TaskCreation task : tasklist) {

System.out.println(task);

}

}

}

public static List<TaskCreation> getTasks() {

return tasklist;

}

}

**Mark Complete Task**

**Mark Complete**

package MarkCompleteTask;

import TaskCreation.TaskCreation;

import java.util.List;

import java.util.Scanner;

public class MarkComplete {

private static final String[] motivationalQuotes = {

"Keep going, you’re doing great!",

"Success is the sum of small efforts repeated daily.",

"Every accomplishment starts with the decision to try.",

"Stay positive, work hard, make it happen.",

"Don't watch the clock; do what it does. Keep going.",

"Believe you can and you’re halfway there.",

"The harder you work for something, the greater you’ll feel when you achieve it.",

"Dream it. Wish it. Do it.",

"Don’t stop when you’re tired. Stop when you’re done.",

"Success doesn’t come to you — you go to it.",

"The only limit to our realization of tomorrow is our doubts of today.",

"Push yourself, because no one else is going to do it for you.",

"Great things never come from comfort zones.",

"Don’t wait for opportunity. Create it.",

"Sometimes we’re tested not to show our weaknesses, but to discover our strengths.",

"Hard work beats talent when talent doesn’t work hard.",

"Your limitation—it's only your imagination.",

"Success is not final, failure is not fatal: It is the courage to continue that counts.",

"You don’t have to be perfect to be amazing.",

"If it doesn’t challenge you, it won’t change you.",

"Work hard in silence, let success make the noise.",

"Don’t wish for it, work for it.",

"You are capable of amazing things.",

"Sometimes the smallest step in the right direction ends up being the biggest step of your life.",

"Don’t be afraid to give up the good to go for the great.",

"The future depends on what you do today.",

"What you get by achieving your goals is not as important as what you become by achieving your goals.",

"Success is the result of preparation, hard work, and learning from failure.",

"Little by little, a little becomes a lot.",

"The difference between ordinary and extraordinary is that little extra.",

"Don’t limit your challenges. Challenge your limits.",

"You don’t have to see the whole staircase, just take the first step.",

"Do something today that your future self will thank you for.",

"Start where you are. Use what you have. Do what you can.",

"The best time to plant a tree was 20 years ago. The second best time is now.",

"Difficult roads often lead to beautiful destinations.",

"Believe in yourself and all that you are.",

"Your only limit is you.",

"Success is walking from failure to failure with no loss of enthusiasm.",

"Motivation is what gets you started. Habit is what keeps you going.",

"Don’t count the days, make the days count.",

"Act as if what you do makes a difference. It does.",

"Never stop doing your best just because someone doesn’t give you credit.",

"There is no substitute for hard work.",

"You are stronger than you think.",

"Don’t watch the clock; do what it does. Keep going.",

"Don’t be pushed around by the fears in your mind. Be led by the dreams in your heart.",

"Success isn’t overnight. It’s when every day you get a little better than the day before.",

"The secret of getting ahead is getting started.",

"Don’t let yesterday take up too much of today.",

"Failure is the condiment that gives success its flavor.",

"Strive for progress, not perfection.",

"Every day may not be good, but there is something good in every day.",

"The best way to get started is to quit talking and begin doing.",

"You miss 100% of the shots you don’t take.",

"Don’t let what you cannot do interfere with what you can do.",

"To be successful, you must accept all challenges that come your way.",

"Work until your idols become your rivals.",

"Your dreams don’t work unless you do.",

"Don’t stop until you’re proud.",

"Don’t quit. Suffer now and live the rest of your life as a champion.",

"Success is liking yourself, liking what you do, and liking how you do it.",

"The only way to do great work is to love what you do.",

"Success usually comes to those who are too busy to be looking for it.",

"It always seems impossible until it’s done.",

"The harder the battle, the sweeter the victory.",

"If you want it, work for it.",

"Don’t wait for the perfect moment, take the moment and make it perfect.",

"Success is not in what you have, but who you are.",

"You don’t find willpower, you create it.",

"The pain you feel today will be the strength you feel tomorrow.",

"Great things take time. Be patient and keep pushing.",

"Sometimes later becomes never. Do it now.",

"Your attitude determines your direction.",

"Keep your face always toward the sunshine—and shadows will fall behind you.",

"The difference between a successful person and others is not a lack of strength, not a lack of knowledge, but rather a lack in will.",

"Don’t be afraid to fail. Be afraid not to try.",

"You were born to be real, not perfect.",

"Start where you are. Use what you have. Do what you can.",

"Don’t stop believing.",

"What lies behind us and what lies before us are tiny matters compared to what lies within us.",

"Success is no accident. It is hard work, perseverance, learning, studying, sacrifice and most of all, love of what you are doing.",

"Work hard, stay positive, and get up early. It’s the best part of the day.",

"The best revenge is massive success.",

"Difficulties in life are intended to make us better, not bitter.",

"Keep your eyes on the stars and your feet on the ground.",

"The best way out is always through.",

"Don’t let fear decide your future.",

"If opportunity doesn’t knock, build a door.",

"The difference between try and triumph is a little “umph.”",

"Keep moving forward.",

"Don’t be discouraged. It’s often the last key in the bunch that opens the lock.",

"Your life does not get better by chance, it gets better by change.",

"You are braver than you believe, stronger than you seem, and smarter than you think.",

"Success is the progressive realization of a worthy goal or ideal.",

"Every strike brings me closer to the next home run.",

"Don’t give up. Great things take time.",

"You only fail when you stop trying.",

"Believe in your infinite potential. Your only limitations are those you set upon yourself.",

"Don’t be afraid to start over. It’s a new chance to rebuild what you want."

};

public static void markCompleteOrReopen(List<TaskCreation> tasklist) {

Scanner scanner = new Scanner(System.in);

if (tasklist.isEmpty()) {

System.out.println("No tasks available to update.");

return;

}

System.out.println("\nTasks:");

for (TaskCreation task : tasklist) {

System.out.printf("ID: %d | Name: %s | Completed: %s\n",

task.getTaskId(), task.getTask\_name(), task.isComplete() ? "Yes" : "No");

}

System.out.print("Enter Task ID to update completion status: ");

int id = Integer.parseInt(scanner.nextLine());

TaskCreation selectedTask = null;

for (TaskCreation task : tasklist) {

if (task.getTaskId() == id) {

selectedTask = task;

break;

}

}

if (selectedTask == null) {

System.out.println("No task found with ID: " + id);

return;

}

if (selectedTask.isComplete()) {

System.out.print("Task is already complete. Do you want to reopen it? (yes/no): ");

String reopen = scanner.nextLine().trim().toLowerCase();

if (reopen.equals("yes") || reopen.equals("y")) {

selectedTask.setComplete(false);

System.out.println("Task reopened successfully.");

} else {

System.out.println("Task status unchanged.");

}

} else {

selectedTask.setComplete(true);

System.out.println("Task marked as complete.");

int index = (int) (Math.random() \* motivationalQuotes.length);

System.out.println("💡 Motivation: " + motivationalQuotes[index]);

}

}

}

**Notes Manager**

**Add Note**

package NotesManager;

import java.util.List;

import java.util.Scanner;

public class AddNote {

private List<Note> noteList;

private Scanner scanner;

public AddNote(List<Note> noteList) {

this.noteList = noteList;

this.scanner = new Scanner(System.in);

}

public void add() {

boolean continueAdding = true;

while (continueAdding) {

String title = "";

while (title.trim().isEmpty()) {

System.out.print("Enter Note Title: ");

title = scanner.nextLine();

if (title.trim().isEmpty()) {

System.out.println("Title cannot be empty!");

}

}

System.out.println("Enter Note Content (type 'END' on a new line to finish):");

StringBuilder contentBuilder = new StringBuilder();

while (true) {

String line = scanner.nextLine();

if (line.equalsIgnoreCase("END")) {

break;

}

contentBuilder.append(line).append("\n");

}

String content = contentBuilder.toString().trim();

Note newNote = new Note(title, content);

noteList.add(newNote);

System.out.println("" +

"Note added with ID: " + newNote.getNoteId());

System.out.print("Add another note? (yes/no): ");

String choice = scanner.nextLine();

if (!choice.equalsIgnoreCase("yes")) {

continueAdding = false;

}

}

}

}

**Delete Note**

package NotesManager;

import java.util.List;

import java.util.Scanner;

public class AddNote {

private List<Note> noteList;

private Scanner scanner;

public AddNote(List<Note> noteList) {

this.noteList = noteList;

this.scanner = new Scanner(System.in);

}

public void add() {

boolean continueAdding = true;

while (continueAdding) {

String title = "";

while (title.trim().isEmpty()) {

System.out.print("Enter Note Title: ");

title = scanner.nextLine();

if (title.trim().isEmpty()) {

System.out.println("Title cannot be empty!");

}

}

System.out.println("Enter Note Content (type 'END' on a new line to finish):");

StringBuilder contentBuilder = new StringBuilder();

while (true) {

String line = scanner.nextLine();

if (line.equalsIgnoreCase("END")) {

break;

}

contentBuilder.append(line).append("\n");

}

String content = contentBuilder.toString().trim();

Note newNote = new Note(title, content);

noteList.add(newNote);

System.out.println("" +

"Note added with ID: " + newNote.getNoteId());

System.out.print("Add another note? (yes/no): ");

String choice = scanner.nextLine();

if (!choice.equalsIgnoreCase("yes")) {

continueAdding = false;

}

}

}

}

**Display Note**

package NotesManager;

import java.util.List;

import java.util.Scanner;

public class AddNote {

private List<Note> noteList;

private Scanner scanner;

public AddNote(List<Note> noteList) {

this.noteList = noteList;

this.scanner = new Scanner(System.in);

}

public void add() {

boolean continueAdding = true;

while (continueAdding) {

String title = "";

while (title.trim().isEmpty()) {

System.out.print("Enter Note Title: ");

title = scanner.nextLine();

if (title.trim().isEmpty()) {

System.out.println("Title cannot be empty!");

}

}

System.out.println("Enter Note Content (type 'END' on a new line to finish):");

StringBuilder contentBuilder = new StringBuilder();

while (true) {

String line = scanner.nextLine();

if (line.equalsIgnoreCase("END")) {

break;

}

contentBuilder.append(line).append("\n");

}

String content = contentBuilder.toString().trim();

Note newNote = new Note(title, content);

noteList.add(newNote);

System.out.println("" +

"Note added with ID: " + newNote.getNoteId());

System.out.print("Add another note? (yes/no): ");

String choice = scanner.nextLine();

if (!choice.equalsIgnoreCase("yes")) {

continueAdding = false;

}

}

}

}

**Note**

package NotesManager;

import java.util.List;

import java.util.Scanner;

public class AddNote {

private List<Note> noteList;

private Scanner scanner;

public AddNote(List<Note> noteList) {

this.noteList = noteList;

this.scanner = new Scanner(System.in);

}

public void add() {

boolean continueAdding = true;

while (continueAdding) {

String title = "";

while (title.trim().isEmpty()) {

System.out.print("Enter Note Title: ");

title = scanner.nextLine();

if (title.trim().isEmpty()) {

System.out.println("Title cannot be empty!");

}

}

System.out.println("Enter Note Content (type 'END' on a new line to finish):");

StringBuilder contentBuilder = new StringBuilder();

while (true) {

String line = scanner.nextLine();

if (line.equalsIgnoreCase("END")) {

break;

}

contentBuilder.append(line).append("\n");

}

String content = contentBuilder.toString().trim();

Note newNote = new Note(title, content);

noteList.add(newNote);

System.out.println("" +

"Note added with ID: " + newNote.getNoteId());

System.out.print("Add another note? (yes/no): ");

String choice = scanner.nextLine();

if (!choice.equalsIgnoreCase("yes")) {

continueAdding = false;

}

}

}

}

**Notes Manager**

package NotesManager;

import java.util.\*;

public class NotesManager {

private List<Note> notes;

private AddNote addNote;

private SearchNote searchNote;

private DisplayNote displayNote;

private DeleteNote deleteNote;

private Scanner scanner;

public NotesManager() {

notes = new ArrayList<>();

addNote = new AddNote(notes);

searchNote = new SearchNote(notes);

displayNote = new DisplayNote(notes);

deleteNote = new DeleteNote(notes);

scanner = new Scanner(System.in);

}

private void runNotesManager() {

boolean exit = false;

while (!exit) {

System.out.println("\nNotes Manager:");

System.out.println("1. Add Note");

System.out.println("2. Search Note by ID");

System.out.println("3. Search Note by Title");

System.out.println("4. Display All Notes");

System.out.println("5. Delete Note by ID");

System.out.println("6. Delete Note by Title");

System.out.println("0. Exit");

System.out.print("Enter choice: ");

int choice = scanner.nextInt();

scanner.nextLine();

switch (choice) {

case 1 -> addNote.add();

case 2 -> searchNote.searchById();

case 3 -> searchNote.searchByTitle();

case 4 -> displayNote.displayAll();

case 5 -> deleteNote.deleteById();

case 6 -> deleteNote.deleteByTitle();

case 0 -> exit = true;

default -> System.out.println("Invalid choice. Try again.");

}

}

}

public static void showNotesMenu() {

NotesManager manager = new NotesManager();

manager.runNotesManager();

}

}

**Search Note**

package NotesManager;

import java.util.List;

import java.util.Scanner;

public class SearchNote {

private List<Note> noteList;

private Scanner scanner;

public SearchNote(List<Note> noteList) {

this.noteList = noteList;

this.scanner = new Scanner(System.in);

}

public void searchById() {

System.out.print("Enter Note ID to search: ");

int id = scanner.nextInt();

scanner.nextLine();

for (Note note : noteList) {

if (note.getNoteId() == id) {

System.out.println("Title: " + note.getTitle());

System.out.println("Content: " + note.getContent());

return;

}

}

System.out.println("Note not found.");

}

public void searchByTitle() {

System.out.print("Enter Note Title to search: ");

String title = scanner.nextLine();

for (Note note : noteList) {

if (note.getTitle().equalsIgnoreCase(title)) {

System.out.println("ID: " + note.getNoteId());

System.out.println("Content: " + note.getContent());

return;

}

}

System.out.println("Note not found.");

}

}

**Edit Or Delete**

**Delete**

package EditOrDelete;

import TaskCreation.TaskCreationMain;

import TaskCreation.TaskCreation;

import TaskCreation.Assignment\_task;

import TaskCreation.Exam\_task;

import TaskCreation.General\_task;

import java.util.List;

import java.util.Scanner;

public class Delete {

public static void deleteTask(List<TaskCreation> tasklist) {

Scanner scanner = new Scanner(System.in);

if (tasklist.isEmpty()) {

System.out.println("No tasks to delete.");

return;

}

System.out.println("Available Tasks:");

for (TaskCreation task : tasklist) {

System.out.println("ID: " + task.getTaskId() + " | Name: " + task.getTask\_name());

}

System.out.print("Enter the ID of the task to delete: ");

int idToDelete;

try {

idToDelete = Integer.parseInt(scanner.nextLine());

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a numeric ID.");

return;

}

TaskCreation taskToRemove = null;

for (TaskCreation task : tasklist) {

if (task.getTaskId()==idToDelete) {

taskToRemove = task;

break;

}

}

if (taskToRemove != null) {

tasklist.remove(taskToRemove);

System.out.println("Task deleted successfully.");

} else {

System.out.println("No task found with ID: " + idToDelete);

}

}

}

**Edit**

**package EditOrDelete;**

import TaskCreation.TaskCreation;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

import java.util.Scanner;

public class Edit {

public static void editTask(List<TaskCreation> tasklist) {

Scanner scanner = new Scanner(System.in);

if (tasklist.isEmpty()) {

System.out.println("No tasks to edit.");

return;

}

System.out.println("\nAvailable Tasks:");

for (TaskCreation task : tasklist) {

System.out.println("ID: " + task.getTaskId() + " | Name: " + task.getTask\_name());

}

System.out.print("\nEnter the ID of the task you want to edit: ");

int inputId = scanner.nextInt();

scanner.nextLine();

TaskCreation selectedTask = null;

for (TaskCreation task : tasklist) {

if (task.getTaskId() == inputId) {

selectedTask = task;

break;

}

}

if (selectedTask == null) {

System.out.println("No task found with ID: " + inputId);

return;

}

System.out.print("Enter new task name (leave empty to keep \"" + selectedTask.getTask\_name() + "\"): ");

String newName = scanner.nextLine();

if (!newName.trim().isEmpty()) {

selectedTask.setTask\_name(newName.trim());

}

System.out.print("Enter new due date and time (dd-MM-yyyy hh:mm AM/PM) or leave empty to keep current: ");

String dateInput = scanner.nextLine();

if (!dateInput.trim().isEmpty()) {

Date newDate = parseDate(dateInput.trim());

if (newDate != null) {

selectedTask.setDue\_date(newDate);

}

}

System.out.print("Enter new priority (leave empty to keep \"" + selectedTask.getPriority() + "\"): ");

String newPriority = scanner.nextLine();

if (!newPriority.trim().isEmpty()) {

selectedTask.setPriority(newPriority.trim());

}

System.out.print("Mark as complete? (true/false) or leave empty to keep \"" + selectedTask.isComplete() + "\": ");

String completeInput = scanner.nextLine();

if (!completeInput.trim().isEmpty()) {

selectedTask.setComplete(Boolean.parseBoolean(completeInput.trim()));

}

System.out.println("\n✅ Task updated successfully!");

}

private static Date parseDate(String dateString) {

try {

return new SimpleDateFormat("dd-MM-yyyy hh:mm a").parse(dateString);

} catch (Exception e) {

System.out.println("❌ Invalid date format. Please use dd-MM-yyyy hh:mm AM/PM.");

return null;

}

}

}

**Edit Or Delete Main**

package EditOrDelete;

import TaskCreation.TaskCreation;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

import java.util.Scanner;

public class Edit {

public static void editTask(List<TaskCreation> tasklist) {

Scanner scanner = new Scanner(System.in);

if (tasklist.isEmpty()) {

System.out.println("No tasks to edit.");

return;

}

System.out.println("\nAvailable Tasks:");

for (TaskCreation task : tasklist) {

System.out.println("ID: " + task.getTaskId() + " | Name: " + task.getTask\_name());

}

System.out.print("\nEnter the ID of the task you want to edit: ");

int inputId = scanner.nextInt();

scanner.nextLine();

TaskCreation selectedTask = null;

for (TaskCreation task : tasklist) {

if (task.getTaskId() == inputId) {

selectedTask = task;

break;

}

}

if (selectedTask == null) {

System.out.println("No task found with ID: " + inputId);

return;

}

System.out.print("Enter new task name (leave empty to keep \"" + selectedTask.getTask\_name() + "\"): ");

String newName = scanner.nextLine();

if (!newName.trim().isEmpty()) {

selectedTask.setTask\_name(newName.trim());

}

System.out.print("Enter new due date and time (dd-MM-yyyy hh:mm AM/PM) or leave empty to keep current: ");

String dateInput = scanner.nextLine();

if (!dateInput.trim().isEmpty()) {

Date newDate = parseDate(dateInput.trim());

if (newDate != null) {

selectedTask.setDue\_date(newDate);

}

}

System.out.print("Enter new priority (leave empty to keep \"" + selectedTask.getPriority() + "\"): ");

String newPriority = scanner.nextLine();

if (!newPriority.trim().isEmpty()) {

selectedTask.setPriority(newPriority.trim());

}

System.out.print("Mark as complete? (true/false) or leave empty to keep \"" + selectedTask.isComplete() + "\": ");

String completeInput = scanner.nextLine();

if (!completeInput.trim().isEmpty()) {

selectedTask.setComplete(Boolean.parseBoolean(completeInput.trim()));

}

System.out.println("\n✅ Task updated successfully!");

}

private static Date parseDate(String dateString) {

try {

return new SimpleDateFormat("dd-MM-yyyy hh:mm a").parse(dateString);

} catch (Exception e) {

System.out.println("❌ Invalid date format. Please use dd-MM-yyyy hh:mm AM/PM.");

return null;

}

}

}

**Search Task**

**Search**

package SearchTask;

import TaskCreation.TaskCreation;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

import java.util.Scanner;

public class Search {

public static void searchTask(List<TaskCreation> tasklist) {

Scanner scanner = new Scanner(System.in);

SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");

System.out.println("\n🔍 Search Task by:");

System.out.println("1. Task ID");

System.out.println("2. Task Name");

System.out.println("3. Priority");

System.out.println("4. Due Date (dd-MM-yyyy)");

System.out.print("Enter your choice: ");

int choice = Integer.parseInt(scanner.nextLine());

boolean found = false;

switch (choice) {

case 1 -> {

System.out.print("Enter Task ID: ");

int id = Integer.parseInt(scanner.nextLine());

for (TaskCreation task : tasklist) {

if (task.getTaskId() == id) {

System.out.println(task);

found = true;

break;

}

}

}

case 2 -> {

System.out.print("Enter Task Name: ");

String name = scanner.nextLine().trim().toLowerCase();

for (TaskCreation task : tasklist) {

if (task.getTask\_name().toLowerCase().contains(name)) {

System.out.println(task);

found = true;

}

}

}

case 3 -> {

System.out.print("Enter Priority (High/Medium/Low): ");

String priority = scanner.nextLine().trim().toLowerCase();

for (TaskCreation task : tasklist) {

if (task.getPriority().toLowerCase().equals(priority)) {

System.out.println(task);

found = true;

}

}

}

case 4 -> {

System.out.print("Enter Due Date (dd-MM-yyyy): ");

String dateInput = scanner.nextLine().trim();

try {

Date inputDate = sdf.parse(dateInput);

for (TaskCreation task : tasklist) {

if (sdf.format(task.getDue\_date()).equals(dateInput)) {

System.out.println(task);

found = true;

}

}

} catch (Exception e) {

System.out.println("❌ Invalid date format.");

return;

}

}

default -> {

System.out.println("❌ Invalid option.");

return;

}

}

if (!found) {

System.out.println("⚠️ No matching tasks found.");

}

}

}

**POMODORO TIMER**

package PomodoroTimer**;**import java.util.Timer**;**import java.util.TimerTask**;**public class PomodoroTimer {  
  
 private Timer timer**;** private int secondsLeft**;** private int sessionCount = **0;** private boolean isRunning = false**;** private String currentState = "IDLE"**;** private int workDuration**;** private int shortBreakDuration**;** private int longBreakDuration**;** public PomodoroTimer(int workMinutes**,** int shortBreakMinutes**,** int longBreakMinutes) {  
 this.workDuration = workMinutes \* **60;** this.shortBreakDuration = shortBreakMinutes \* **60;** this.longBreakDuration = longBreakMinutes \* **60;** }  
  
 public void start() {  
 if (isRunning) {  
 System.*out*.println("Timer is already running.")**;** return**;** }  
  
  
 if (currentState.equals("SHORT\_BREAK") || currentState.equals("LONG\_BREAK")) {  
 System.*out*.println("You're on a break! Wait for it to finish.")**;** return**;** }  
  
  
 if (currentState.equals("IDLE")) {  
 currentState = "WORK"**;** secondsLeft = workDuration**;** System.*out*.println("Started: " + currentState)**;** }  
  
 isRunning = true**;** timer = new Timer()**;** timer.scheduleAtFixedRate(new TimerTask() {  
 public void run() {  
 if (secondsLeft > **0**) {  
 secondsLeft--**;** showTime()**;** } else {  
 timer.cancel()**;** isRunning = false**;** sessionEnd()**;** }  
 }  
 }**, 0, 1000**)**;** }  
  
 public void stop() {  
 if (timer != null) {  
 timer.cancel()**;** isRunning = false**;** System.*out*.println("Timer stopped.")**;** }  
 }  
  
 public void reset() {  
 stop()**;** secondsLeft = **0;** sessionCount = **0;** currentState = "IDLE"**;** System.*out*.println("Timer reset.")**;** }  
  
 private void sessionEnd() {  
 if (currentState.equals("WORK")) {  
 sessionCount++**;** System.*out*.println("Work session done! (" + sessionCount + " total)")**;** if (sessionCount % **4** == **0**) {  
 currentState = "LONG\_BREAK"**;** secondsLeft = longBreakDuration**;** System.*out*.println("Long break (" + longBreakDuration / **60** + " mins)")**;** } else {  
 currentState = "SHORT\_BREAK"**;** secondsLeft = shortBreakDuration**;** System.*out*.println("Short break (" + shortBreakDuration / **60** + " mins)")**;** }  
  
 start()**;** } else {  
 currentState = "IDLE"**;** System.*out*.println(" Break finished. Ready to start next session.")**;** }  
 }  
  
 private void showTime() {  
 int min = secondsLeft / **60;** int sec = secondsLeft % **60;** System.*out*.printf("⏲️ %02d:%02d [%s]%n"**,** min**,** sec**,** currentState)**;** }  
}

**TIMER MAIN**

package PomodoroTimer**;**import java.util.Scanner**;**public class TimerMian {  
  
 public static void startTimer() {  
 Scanner input = new Scanner(System.*in*)**;** System.*out*.print("Enter work duration (minutes): ")**;** int work = input.nextInt()**;** System.*out*.print("Enter short break duration (minutes): ")**;** int shortBreak = input.nextInt()**;** System.*out*.print("Enter long break duration (minutes): ")**;** int longBreak = input.nextInt()**;** PomodoroTimer timer = new PomodoroTimer(work**,** shortBreak**,** longBreak)**;** while (true) {  
 System.*out*.println("\nChoose: 1. Start 2. Stop 3. Reset 4. Exit")**;** int choice = input.nextInt()**;** switch (choice) {  
 case **1**:  
 timer.start()**;** break**;** case **2**:  
 timer.stop()**;** break**;** case **3**:  
 timer.reset()**;** break**;** case **4**:  
 timer.stop()**;** System.*out*.println("Goodbye!")**;** return**;** default:  
 System.*out*.println("Invalid choice.")**;** }  
 }  
 }  
}

**MOOD SELECTOR**

package MoodSelection**;**import java.util.Scanner**;**public class MoodSelector {  
  
 public static String selectMood() {  
 Scanner scanner = new Scanner(System.*in*)**;** System.*out*.println(" Select Your Current Mood:")**;** System.*out*.println("1. Happy")**;** System.*out*.println("2. Sad")**;** System.*out*.println("3. Angry")**;** System.*out*.println("4. Tired")**;** System.*out*.println("5. Motivated")**;** System.*out*.println("6. Neutral")**;** System.*out*.print("Enter your choice (1-6): ")**;** int choice = scanner.nextInt()**;** String mood = ""**;** switch (choice) {  
 case **1** -> mood = "Happy"**;** case **2** -> mood = "Sad"**;** case **3** -> mood = "Angry"**;** case **4** -> mood = "Tired"**;** case **5** -> mood = "Motivated"**;** case **6** -> mood = "Neutral"**;** default -> {  
 System.*out*.println(" Invalid choice. Please select between 1 to 6.")**;** return " "**;** }  
 }  
  
 System.*out*.println(" You selected mood: " + mood)**;** return mood**;** }  
}

**TASKSUGESTIONMOOD**

package MoodSelection**;**import TaskCreation.TaskCreation**;**import java.util.Date**;**import static TaskCreation.TaskCreationMain.*tasklist***;**public class TaskSuggestionMood {  
 public static void suggestTasksByMood(String mood) {  
 Date now = new Date()**;** boolean found = false**;** System.*out*.println("\n Suggested Tasks for Mood: " + mood)**;** for (TaskCreation task : *tasklist*) {  
 String priority = task.getPriority()**;** Date dueDate = task.getDue\_date()**;** switch (mood.toLowerCase()) {  
 case "motivated" -> {  
 if (priority.equalsIgnoreCase("High") && dueDate.before(now)) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
 case "tired"**,** "sad" -> {  
 long diff = dueDate.getTime() - now.getTime()**;** long hoursLeft = diff / (**1000** \* **60** \* **60**)**;** if ((priority.equalsIgnoreCase("Low") || priority.equalsIgnoreCase("Medium")) && hoursLeft > **24**) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
 case "happy"**,** "neutral" -> {  
 if (priority.equalsIgnoreCase("Medium")) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
 case "angry" -> {  
 long diff = dueDate.getTime() - now.getTime()**;** long hoursLeft = diff / (**1000** \* **60** \* **60**)**;** if (priority.equalsIgnoreCase("Low") && hoursLeft <= **48**) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
 }  
 }  
  
  
 if (!found) {  
 System.*out*.println(" No tasks matched exactly. Showing fallback suggestions:")**;** for (TaskCreation task : *tasklist*) {  
 String priority = task.getPriority()**;** Date dueDate = task.getDue\_date()**;** switch (mood.toLowerCase()) {  
 case "motivated" -> {  
 if (priority.equalsIgnoreCase("Medium") && dueDate.after(now)) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
 case "sad"**,** "tired" -> {  
 if (priority.equalsIgnoreCase("Medium")) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
 case "happy"**,** "neutral" -> {  
 if (dueDate.after(now)) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
 case "angry" -> {  
 long diff = dueDate.getTime() - now.getTime()**;** long hoursLeft = diff / (**1000** \* **60** \* **60**)**;** if (hoursLeft <= **72**) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
 }  
 }  
  
 if (!found) {  
 System.*out*.println(" Still no fallback tasks available for this mood.")**;** }  
 }  
 }  
}

**MOODSELECTORMAIN**

package MoodSelection**;**import java.util.Scanner**;**public class MoodSelectorSugeestionmain {  
  
  
 public static void MoodSelector() {  
 Scanner scanner = new Scanner(System.*in*)**;** while (true) {  
 System.*out*.println("\n Mood-Based Task Suggestion Menu ")**;** System.*out*.println("1. Select Mood and View Suggestions")**;** System.*out*.println("2. Back to Main Menu")**;** System.*out*.print("Enter your choice (1-2): ")**;** String choice = scanner.nextLine()**;** switch (choice) {  
 case "1":  
 String selectedMood = MoodSelector.*selectMood*()**;** if (!selectedMood.isEmpty()) {  
 TaskSuggestionMood.*suggestTasksByMood*(selectedMood)**;** }  
 break**;** case "2":  
 return**;** default:  
 System.*out*.println(" Invalid choice. Please try again.")**;** break**;** }  
  
 }  
 }  
 }

SUMMARY OF TASKS

package SummaryOFTask**;**import TaskCreation.TaskCreation**;**import static TaskCreation.TaskCreationMain.*tasklist***;**public class SummaryOfWork {  
  
 public static void showSummary() {  
 int total = *tasklist*.size()**;** int completed = **0;** int pending = **0;** int highPriority = **0;** for (TaskCreation task : *tasklist*) {  
 if (task.isComplete()) {  
 completed++**;** } else {  
 pending++**;** }  
  
 if (task.getPriority().equalsIgnoreCase("High")) {  
 highPriority++**;** }  
 }  
  
 double consistency = (total > **0**) ? ((double) completed / total) \* **100** : **0;** System.*out*.println(" Summary:")**;** System.*out*.println("Total Tasks: " + total +  
 " | Completed: " + completed +  
 " | Pending: " + pending +  
 " | High Priority: " + highPriority)**;** System.*out*.printf("Consistency: %.2f%%\n"**,** consistency)**;** }  
}

**VIEW TASKS**

**BY DATE**

package ViewTasks**;**import TaskCreation.TaskCreation**;**import java.text.SimpleDateFormat**;**import java.util.\***;**import static TaskCreation.TaskCreationMain.*tasklist***;**public class ByDates {  
  
 public static void filterTasksByDueDate(String option**,** String startDateStr**,** String endDateStr) {  
 boolean found = false**;** Date today = new Date()**;** Calendar cal = Calendar.*getInstance*()**;** for (TaskCreation task : *tasklist*) {  
 Date dueDate = task.getDue\_date()**;** if (option.equalsIgnoreCase("today")) {  
 if (*isSameDay*(dueDate**,** today)) {  
 System.*out*.println(task)**;** found = true**;** }  
 } else if (option.equalsIgnoreCase("week")) {  
 cal.setTime(today)**;** cal.add(Calendar.*DAY\_OF\_YEAR***, 7**)**;** Date endOfWeek = cal.getTime()**;** if (dueDate.after(today) && dueDate.before(endOfWeek)) {  
 System.*out*.println(task)**;** found = true**;** }  
 } else if (option.equalsIgnoreCase("custom")) {  
 try {  
 SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy")**;** Date start = sdf.parse(startDateStr)**;** Date end = sdf.parse(endDateStr)**;** cal.setTime(end)**;** cal.set(Calendar.*HOUR\_OF\_DAY***, 23**)**;** cal.set(Calendar.*MINUTE***, 59**)**;** cal.set(Calendar.*SECOND***, 59**)**;** end = cal.getTime()**;** if (!dueDate.before(start) && !dueDate.after(end)) {  
 System.*out*.println(task)**;** found = true**;** }  
 } catch (Exception e) {  
 System.*out*.println("Invalid date format. Use dd-MM-yyyy.")**;** return**;** }  
 }  
 }  
  
 if (!found) {  
 System.*out*.println("No tasks found for selected due date option.")**;** }  
 }  
  
 private static boolean isSameDay(Date d1**,** Date d2) {  
 Calendar c1 = Calendar.*getInstance*()**;** Calendar c2 = Calendar.*getInstance*()**;** c1.setTime(d1)**;** c2.setTime(d2)**;** return c1.get(Calendar.*YEAR*) == c2.get(Calendar.*YEAR*) &&  
 c1.get(Calendar.*DAY\_OF\_YEAR*) == c2.get(Calendar.*DAY\_OF\_YEAR*)**;** }  
}

**BY PRIORITY**

package ViewTasks**;**import TaskCreation.TaskCreation**;**import static TaskCreation.TaskCreationMain.*tasklist***;**public class PirioritySearch {  
  
  
 public static void searchTaskByPriority(String priority) {  
 boolean found = false**;** for (TaskCreation task : *tasklist*) {  
 if (task.getPriority().equalsIgnoreCase(priority)) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
  
 if (!found) {  
 System.*out*.println("No task found with the priority: " + priority)**;** }  
 }  
  
  
  
  
}

**BY STATUS**

package ViewTasks**;**import TaskCreation.TaskCreation**;**import static TaskCreation.TaskCreationMain.*tasklist***;**public class ByStatus {  
  
 public static void filterByCompletionStatus(String status) {  
 boolean found = false**;** for (TaskCreation task : *tasklist*) {  
 if (status.equalsIgnoreCase("completed") && task.isComplete()) {  
 System.*out*.println(task)**;** found = true**;** } else if (status.equalsIgnoreCase("incomplete") && !task.isComplete()) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
  
 if (!found) {  
 System.*out*.println("No " + status + " tasks found.")**;** }  
 }  
}

**BY NAME**

package ViewTasks;  
  
import TaskCreation.TaskCreation;  
  
import static TaskCreation.TaskCreationMain.*tasklist*;  
  
public class SearchbyName {  
  
 public static void searchTaskByName(String name) {  
 boolean found = false;  
  
 for (TaskCreation task : *tasklist*) {  
 if (task.getTask\_name().equalsIgnoreCase(name)) {  
 System.*out*.println(task);  
 found = true;  
 }  
 }  
  
 if (!found) {  
 System.*out*.println("No task found with the name: " + name);  
 }  
 }  
  
  
  
}

**TASK TYPE**

package ViewTasks**;**import TaskCreation.Assignment\_task**;**import TaskCreation.Exam\_task**;**import TaskCreation.General\_task**;**import TaskCreation.TaskCreation**;**import static TaskCreation.TaskCreationMain.*tasklist***;**public class TaskType {  
  
  
 public static void searchByType(String type) {  
 boolean found = false**;** for (TaskCreation task : *tasklist*) {  
 if (type.equalsIgnoreCase("exam") && task instanceof Exam\_task) {  
 System.*out*.println(task)**;** found = true**;** } else if (type.equalsIgnoreCase("assignment") && task instanceof Assignment\_task) {  
 System.*out*.println(task)**;** found = true**;** } else if (type.equalsIgnoreCase("general") && task instanceof General\_task) {  
 System.*out*.println(task)**;** found = true**;** }  
 }  
  
 if (!found) {  
 System.*out*.println("No tasks found of type: " + type)**;** }  
 }  
}

**VIEW ALL TASKS**

package ViewTasks**;**import TaskCreation.TaskCreation**;**import java.util.List**;**import static TaskCreation.TaskCreationMain.*tasklist***;**public class ViewAllTasks {  
  
 public static void viewAllTasks(List<TaskCreation> taskList) {  
 if (*tasklist*.isEmpty()) {  
 System.*out*.println("No tasks available.")**;** } else {  
 for (TaskCreation task : *tasklist*) {  
 System.*out*.println(task)**;** }  
 }  
 }  
  
}

**VIEW TASK MAIN**

package ViewTasks**;**import java.util.Scanner**;**public class ViewTaskMain {  
  
 public static void showMenu() {  
 Scanner scanner = new Scanner(System.*in*)**;** while (true) {  
 System.*out*.println("\n--- View Task Menu ---")**;** System.*out*.println("1. Search by Name")**;** System.*out*.println("2. Filter by Priority")**;** System.*out*.println("3. Filter by Task Type")**;** System.*out*.println("4. Filter by Due Date (Today/This Week)")**;** System.*out*.println("5. Filter by Completion Status")**;** System.*out*.println("6. Back to Main Menu")**;** System.*out*.print("Enter your choice (1-6): ")**;** String choice = scanner.nextLine()**;** switch (choice) {  
 case "1" -> {  
 System.*out*.print("Enter task name: ")**;** String name = scanner.nextLine()**;** SearchbyName.*searchTaskByName*(name)**;** }  
 case "2" -> {  
 System.*out*.print("Enter priority (High/Medium/Low): ")**;** String priority = scanner.nextLine()**;** PirioritySearch.*searchTaskByPriority*(priority)**;** }  
 case "3" -> {  
 System.*out*.print("Enter task type (Exam/Assignment/General): ")**;** String type = scanner.nextLine()**;** TaskType.*searchByType*(type)**;** }  
 case "4" -> {  
 System.*out*.println("a. Today")**;** System.*out*.println("b. This Week")**;** System.*out*.println("c. Custom Range")**;** System.*out*.print("Choose (a/b/c): ")**;** String dateChoice = scanner.nextLine()**;** if (dateChoice.equalsIgnoreCase("a")) {  
 ByDates.*filterTasksByDueDate*("today"**,** null**,** null)**;** } else if (dateChoice.equalsIgnoreCase("b")) {  
 ByDates.*filterTasksByDueDate*("week"**,** null**,** null)**;** } else if (dateChoice.equalsIgnoreCase("c")) {  
 System.*out*.print("Enter start date (dd-MM-yyyy): ")**;** String start = scanner.nextLine()**;** System.*out*.print("Enter end date (dd-MM-yyyy): ")**;** String end = scanner.nextLine()**;** ByDates.*filterTasksByDueDate*("custom"**,** start**,** end)**;** } else {  
 System.*out*.println("Invalid choice.")**;** }  
 }  
  
 case "5" -> {  
 System.*out*.print("Enter status (Completed/Incomplete): ")**;** String status = scanner.nextLine()**;** ByStatus.*filterByCompletionStatus*(status)**;** }  
 case "6" -> {  
 return**;** }  
 default -> System.*out*.println("Invalid choice. Try again.")**;** }  
 }  
 }  
}

**SUMMARY OF TASKS**

package SummaryOFTask**;**import TaskCreation.TaskCreation**;**import static TaskCreation.TaskCreationMain.*tasklist***;**public class SummaryOfWork {  
  
 public static void showSummary() {  
 int total = *tasklist*.size()**;** int completed = **0;** int pending = **0;** int highPriority = **0;** for (TaskCreation task : *tasklist*) {  
 if (task.isComplete()) {  
 completed++**;** } else {  
 pending++**;** }  
  
 if (task.getPriority().equalsIgnoreCase("High")) {  
 highPriority++**;** }  
 }  
  
 double consistency = (total > **0**) ? ((double) completed / total) \* **100** : **0;** System.*out*.println(" Summary:")**;** System.*out*.println("Total Tasks: " + total +  
 " | Completed: " + completed +  
 " | Pending: " + pending +  
 " | High Priority: " + highPriority)**;** System.*out*.printf("Consistency: %.2f%%\n"**,** consistency)**;** }

S

**}NOTIFICATION**

package Notifications**;**import TaskCreation.TaskCreation**;**import java.awt.\***;**import java.awt.image.BufferedImage**;**import java.util.\***;**import static TaskCreation.TaskCreationMain.*tasklist***;**public class TaskReminderScheduler {  
  
 private static final Set<String> *notifiedTasks* = new HashSet<>()**;** public static void startReminderCheck() {  
 Timer timer = new Timer()**;** TimerTask reminderTask = new TimerTask() {  
 @Override  
 public void run() {  
 Date now = new Date()**;** for (TaskCreation task : *tasklist*) {  
 if (task.isComplete()) continue**;** Date dueDate = task.getDue\_date()**;** long diff = dueDate.getTime() - now.getTime()**;** long minutesLeft = diff / (**1000** \* **60**)**;** int[] reminderTimes = {**30, 15, 5, 1**}**;** for (int reminderTime : reminderTimes) {  
 if (minutesLeft == reminderTime) {  
 String taskKey = task.getTask\_name() + "\_" + reminderTime + "\_" + dueDate.getTime()**;** if (!*notifiedTasks*.contains(taskKey)) {  
 *showWindowsNotification*(" Task Reminder"**,** " \"" + task.getTask\_name() + "\" is due in " + reminderTime + " minute(s)!")**;** *notifiedTasks*.add(taskKey)**;** }  
 }  
 }  
 }  
 }  
 }**;** timer.scheduleAtFixedRate(reminderTask**, 0, 10** \* **1000**)**;** }  
  
 private static void showWindowsNotification(String title**,** String message) {  
 if (!SystemTray.*isSupported*()) {  
 System.*out*.println("System tray not supported on this system.")**;** return**;** }  
  
 try {  
 SystemTray tray = SystemTray.*getSystemTray*()**;** Image image = new BufferedImage(**16, 16,** BufferedImage.*TYPE\_INT\_ARGB*)**;** TrayIcon trayIcon = new TrayIcon(image**,** "Task Reminder")**;** trayIcon.setImageAutoSize(true)**;** trayIcon.setToolTip("Upcoming Task Notification")**;** tray.add(trayIcon)**;** trayIcon.displayMessage(title**,** message**,** TrayIcon.MessageType.*INFO*)**;** Thread.*sleep*(**150000**)**;** tray.remove(trayIcon)**;** } catch (Exception e) {  
 e.printStackTrace()**;** }  
 }  
}

**LOGIN SIGNUP**

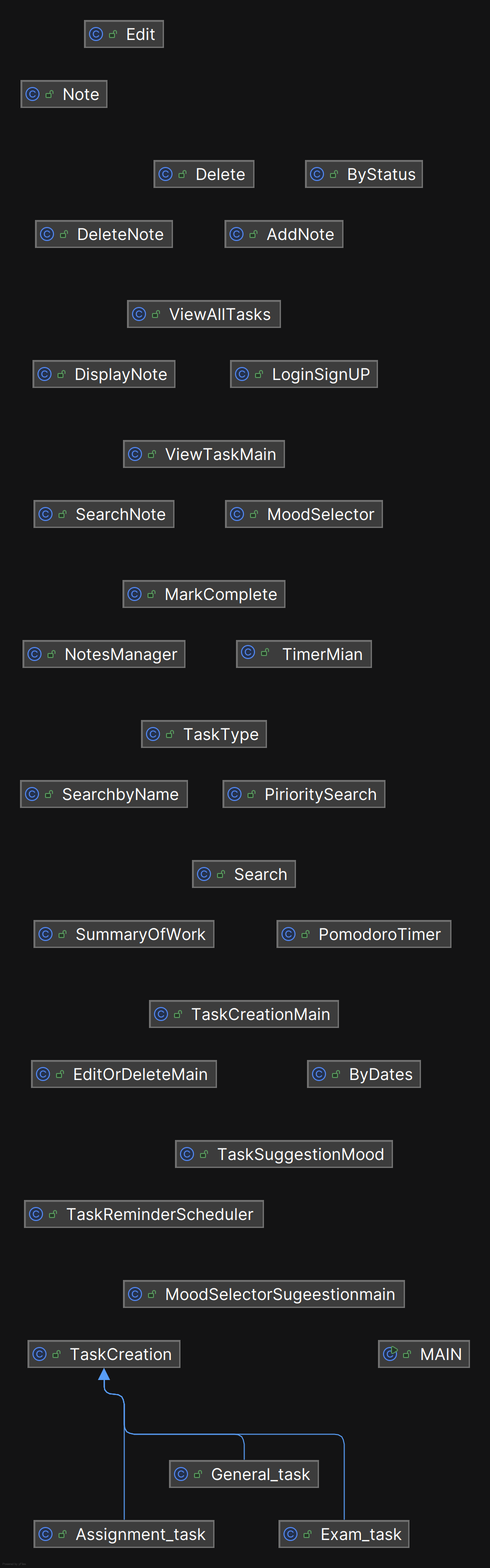
package LoginSignUp**;**import java.sql.\***;**import java.util.Scanner**;**public class LoginSignUP {  
  
 static final String *DB\_URL* = "jdbc:sqlite:users.db"**;** public static Connection connect() {  
 Connection conn = null**;** try {  
 conn = DriverManager.*getConnection*(*DB\_URL*)**;** } catch (SQLException e) {  
 System.*out*.println("Database connection error: " + e.getMessage())**;** }  
 return conn**;** }  
  
  
 public static void createTable() {  
 String sql = "CREATE TABLE IF NOT EXISTS users ("  
 + "id INTEGER PRIMARY KEY AUTOINCREMENT,"  
 + "username TEXT UNIQUE,"  
 + "password TEXT"  
 + ");"**;** try (Connection conn = *connect*()**;** Statement stmt = conn.createStatement()) {  
 stmt.execute(sql)**;** System.*out*.println("Users table ready.")**;** } catch (SQLException e) {  
 System.*out*.println("Table creation error: " + e.getMessage())**;** }  
 }  
  
  
 public static boolean signUp(String username**,** String password) {  
 String sql = "INSERT INTO users(username, password) VALUES(?, ?)"**;** try (Connection conn = *connect*()**;** PreparedStatement pstmt = conn.prepareStatement(sql)) {  
 pstmt.setString(**1,** username)**;** pstmt.setString(**2,** password)**;** pstmt.executeUpdate()**;** System.*out*.println("Signup successful!")**;** return true**;** } catch (SQLException e) {  
 System.*out*.println("Signup failed: Username already taken or error.")**;** return false**;** }  
 }  
  
  
 public static boolean login(String username**,** String password) {  
 String sql = "SELECT \* FROM users WHERE username = ? AND password = ?"**;** try (Connection conn = *connect*()**;** PreparedStatement pstmt = conn.prepareStatement(sql)) {  
 pstmt.setString(**1,** username)**;** pstmt.setString(**2,** password)**;** ResultSet rs = pstmt.executeQuery()**;** if (rs.next()) {  
 System.*out*.println("Login successful! Welcome " + username)**;** return true**;** } else {  
 System.*out*.println("Login failed: Incorrect username or password.")**;** return false**;** }  
 } catch (SQLException e) {  
 System.*out*.println("Login error: " + e.getMessage())**;** return false**;** }  
 }  
  
  
 public static boolean runAuthMenu() {  
 *createTable*()**;** // Make sure users table exists  
 Scanner sc = new Scanner(System.*in*)**;** while (true) {  
 System.*out*.println("\n==== Welcome to Smart Academic Planner ====")**;** System.*out*.println("1. Signup")**;** System.*out*.println("2. Login")**;** System.*out*.println("3. Exit")**;** System.*out*.print("Enter choice: ")**;** String choice = sc.nextLine()**;** switch (choice) {  
 case "1" -> {  
 System.*out*.print("Enter new username: ")**;** String username = sc.nextLine()**;** System.*out*.print("Enter new password: ")**;** String password = sc.nextLine()**;** *signUp*(username**,** password)**;** }  
 case "2" -> {  
 System.*out*.print("Enter username: ")**;** String username = sc.nextLine()**;** System.*out*.print("Enter password: ")**;** String password = sc.nextLine()**;** if (*login*(username**,** password)) {  
 return true**;** }  
  
 }  
 case "3" -> {  
 System.*out*.println("Exiting application. Goodbye!")**;** return false**;** }  
 default -> System.*out*.println("Invalid choice, please try again.")**;** }  
 }  
 }  
  
  
  
}

**Smart Academic Planner Main**

**Main**

package SmartAcademicPlannerMAIN**;**import LoginSignUp.LoginSignUP**;**import MoodSelection.MoodSelectorSugeestionmain**;**import Notifications.TaskReminderScheduler**;**import PomodoroTimer.TimerMian**;**import SummaryOFTask.SummaryOfWork**;**import ViewTasks.ViewTaskMain**;**import EditOrDelete.EditOrDeleteMain**;**import SearchTask.Search**;**import TaskCreation.TaskCreationMain**;**import TaskCreation.TaskCreation**;**import NotesManager.NotesManager**;**import java.util.List**;**import java.util.Scanner**;**public class MAIN {  
  
 public static void main(String[] args) {  
 TaskReminderScheduler.*startReminderCheck*()**;** Scanner scanner = new Scanner(System.*in*)**;** boolean loggedIn = LoginSignUP.*runAuthMenu*()**;** if (!loggedIn) {  
 System.*out*.println("Goodbye!")**;** scanner.close()**;** return**;** }  
  
 boolean running = true**;** while (running) {  
 *clearConsole*()**;** System.*out*.println("\n==== Smart Academic Planner ====")**;** System.*out*.println("1. Task Creation")**;** System.*out*.println("2. Task Edit or Delete")**;** System.*out*.println("3. Task Searcher")**;** System.*out*.println("4. Notes")**;** System.*out*.println("5. View Tasks ")**;** System.*out*.println("6. Mood Based Suggestions")**;** System.*out*.println("7. Summary of Tasks")**;** System.*out*.println("8. Pomodoro Timer")**;** System.*out*.println("9. AI Study Assistant ")**;** System.*out*.println("0. Exit")**;** System.*out*.print("Enter your choice: ")**;** int choice = *getValidatedChoice*(scanner)**;** switch (choice) {  
 case **1** -> *runTaskCreationModule*(TaskCreationMain.*tasklist***,** scanner)**;** case **2** -> EditOrDeleteMain.*showEditOrDeleteMenu*(TaskCreationMain.*tasklist***,** scanner)**;** case **3** -> Search.*searchTask*(TaskCreationMain.*tasklist*)**;** case **4** -> NotesManager.*showNotesMenu*()**;** case **5** -> ViewTaskMain.*showMenu*()**;** case **6** -> MoodSelectorSugeestionmain.*MoodSelector*()**;** case **7** -> SummaryOfWork.*showSummary*()**;** case **8** -> TimerMian.*startTimer*()**;** case **9** -> System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*AI Coming Soon\*\*\*\*\*\*\*\*\* ")**;** case **0** -> {  
 running = false**;** System.*out*.println("Exiting Smart Academic Planner. Goodbye!")**;** }  
 default -> System.*out*.println("Invalid choice. Please try again.")**;** }  
  
 if (running) {  
 System.*out*.print("\nPress Enter to continue...")**;** scanner.nextLine()**;** }  
 }  
  
 scanner.close()**;** }  
  
 public static void runTaskCreationModule(List<TaskCreation> tasklist**,** Scanner scanner) {  
 boolean inTaskModule = true**;** while (inTaskModule) {  
 *clearConsole*()**;** TaskCreationMain.*displayMenu*()**;** System.*out*.println("6. Return to Main Menu")**;** System.*out*.print("Choose an option: ")**;** int subChoice = *getValidatedChoice*(scanner)**;** if (subChoice == **6**) {  
 inTaskModule = false**;** } else {  
 TaskCreationMain.*handleUserInput*(subChoice**,** tasklist)**;** }  
  
 if (inTaskModule) {  
 System.*out*.print("\nPress Enter to continue...")**;** scanner.nextLine()**;** }  
 }  
 }  
  
 private static int getValidatedChoice(Scanner scanner) {  
 String input = scanner.nextLine()**;** try {  
 return Integer.*parseInt*(input)**;** } catch (NumberFormatException e) {  
 return -**1;** }  
 }  
  
 private static void clearConsole() {  
 try {  
 if (System.*getProperty*("os.name").contains("Windows")) {  
 new ProcessBuilder("cmd"**,** "/c"**,** "cls").inheritIO().start().waitFor()**;** } else {  
 System.*out*.print("\033[H\033[2J")**;** System.*out*.flush()**;** }  
 } catch (Exception e) {  
  
 }  
 }  
}

**UML Diagram**

****