Technical Report

WIX1002 Fundamentals of Programming

Project 3: To Do List

Tutorial Group: Group 3

Group SOUT

Lecturer Name:Dr. Mohamad Hazim bin Md Hanif

Group Member:

NO.	NAME	MATRIC NO.
1	Lai Zhi Hang	24066429
2	Gianina Lazaroo	24066775
3	Kwek Chee Ling	23080328
4	Lee Yu Xuan	23098783
5	Mah Kah Mun	24004590

Table of contents

1. **Introduction for To-do list App**

2. **Basic Requirement of To-do list App**
    - Task Creation
    - Task Management
    - Task Deletion
    - Task Sorting
    - Task Searching
    - Recurring Tasks
    - Task Dependencies
    - Edit Task
    - Storage System
    - Data Load State

3. **Extra features**
    - Graphical User Interface
    - Email Notification System
    - Data Analytics

4. **Approach Taken to Solve the Task**

5. **Detailed Description of the Solution**
    - Modules
    - Description with flowchart

6. **Sample Input and Output**

1. Introduction for To-do list App

A to-do list is one of the most popular introductory projects, widely used to demonstrate understanding of key programming concepts and build problem-solving skills. This project offers an opportunity to practice Java programming, object-oriented principles, and basic data storage solutions while working on a real-world application.

Time management is crucial for any student, especially for freshmen who are adjusting to university life. The goal of this project is to develop a simple To-Do List Application that will help students track and manage their tasks effectively. By building this project, students will learn key programming concepts while also creating a useful tool.

2. Basic Requirement of To-do list App

a. **Task Creation**

   Users are able to create tasks with a:

   - Title
   - Description
   - Due Date
   - Completion Status (complete / incomplete)
   - Category (homework / personal / work)
   - Priority Level (low / medium / high)

b. **Task Management**

   Users are able to mark tasks as completed and view task list.

c. **Task Deletion**

   Users are allowed to delete tasks from the checklist.

d. **Task Sorting**

   Users can sort tasks by due date or priority level in either ascending or descending order. The collection algorithm is used to organize tasks efficiently.

e. **Task Searching**

Users can search for relevant tasks by entering keywords that match the task title or description. A Linear Search algorithm is implemented to enable this full-text search functionality.

f. **Recurring Tasks**

Users can add recurring tasks on a daily, weekly, or monthly basis. The system automatically generates new tasks once a recurrence is completed.

g. **Task Dependencies**

Users are allowed to link tasks, marking one as dependent on another. Multiple dependencies and nested dependencies are possible, but cycles of dependencies are not allowed.

h. **Edit Task**

Users are able to select which task to edit for the information.

i. **Storage System**

Tasks are stored in a database implemented using a CSV file.

j. **Data Load State**

The fetched data should be converted and loaded as an object. The system automatically fetches and loads task data from the CSV file when users launch the program.

3. Extra Features of To-do list App

A. Graphical User Interface

Create a simple GUI using JavaFX.

This GUI enables users to:

- Add new tasks.
- View tasks in a list.
- Sort tasks.
- Mark tasks as complete.
- Delete tasks

B. **Email Notification System**

Implement an email notification system that alerts users when a task is due within 24 hours using JavaMail. Users are allowed to input their email address to receive the email notification.

C. **Data Analytics**

Implement a simple analytics dashboard to show the users the following statistics:

- Number of tasks completed vs. pending.
- Task completion rate over time.
- Categorized task summary.

4. Approach Taken to Solve the Task

To implement the To-Do List Application, we employed a modular strategy by dividing the overall functionality into smaller, manageable components. Each feature was developed independently to ensure clarity and accuracy before integrating it into the application. This systematic approach enabled us to:

Analyze Requirements:

Thoroughly assess the project objectives to identify key and optional features.

Design the application's structure and workflow in detail.

Break Down into Modules:

Each feature (e.g., task creation, sorting, recurring tasks) was implemented as a separate function.

Isolated functions simplified debugging and testing processes.

Iterative Development:

Constructed and validated each function individually (e.g., addTask, editTask, sortTasks) before merging them.

Tested with mock data and scenarios to confirm functionality.

Integration and Comprehensive Testing:

Combined the individual functions into the main program while preserving modularity.
Validated seamless interaction among modules and conducted extensive edge-case testing.

This structured approach ensured the application was robust and maintainable, enabling the addition of advanced features like a GUI, email notifications, and data analytics without affecting the core functionality.

5. Detailed Description of the Solution

Modules

1.Task Management Module

Features: Task creation, marking tasks as complete/incomplete, editing, deleting tasks and view tasks.

2. Task Sorting Module

Features: Sorting tasks based on due date or priority using sorting algorithms like Bubble Sort or Selection Sort.

3.Task Searching Module

Features: Search for tasks using a keyword in title or description with algorithms like Linear or Binary Search.

4. Recurring Tasks Module

Features: Automate task creation for daily, weekly, or monthly recurrence.

5.Task Dependency Module

Features: Establish dependencies among tasks and enforce completion rules.

6. Data Storage and Loading Module

Features: Store tasks in a database or CSV and load them into the application on startup.

7. Graphical User Interface

Features: GUI implementation using JavaFX or another tool for managing tasks visually.

Details: Screenshots of the GUI and a description of its functionality.

Advanced Features Module (if implemented)

8. Email Notification System

Features:Send task reminders.

9. Data Analytics Dashboard

Features: Task statistics and insights.

10. Error Handling and Validation

Features: Manage user errors like invalid inputs, task overlaps, etc.

Description with flowchart

Basic Features

1. Create a Task Class

The SimpleTaskManager Class provides a command-line interface for managing tasks. It supports adding, marking, deleting, sorting, and viewing tasks, as well as handling recurring tasks. Tasks are stored as an ArrayList of String[] arrays, with each task's details encapsulated within the array.

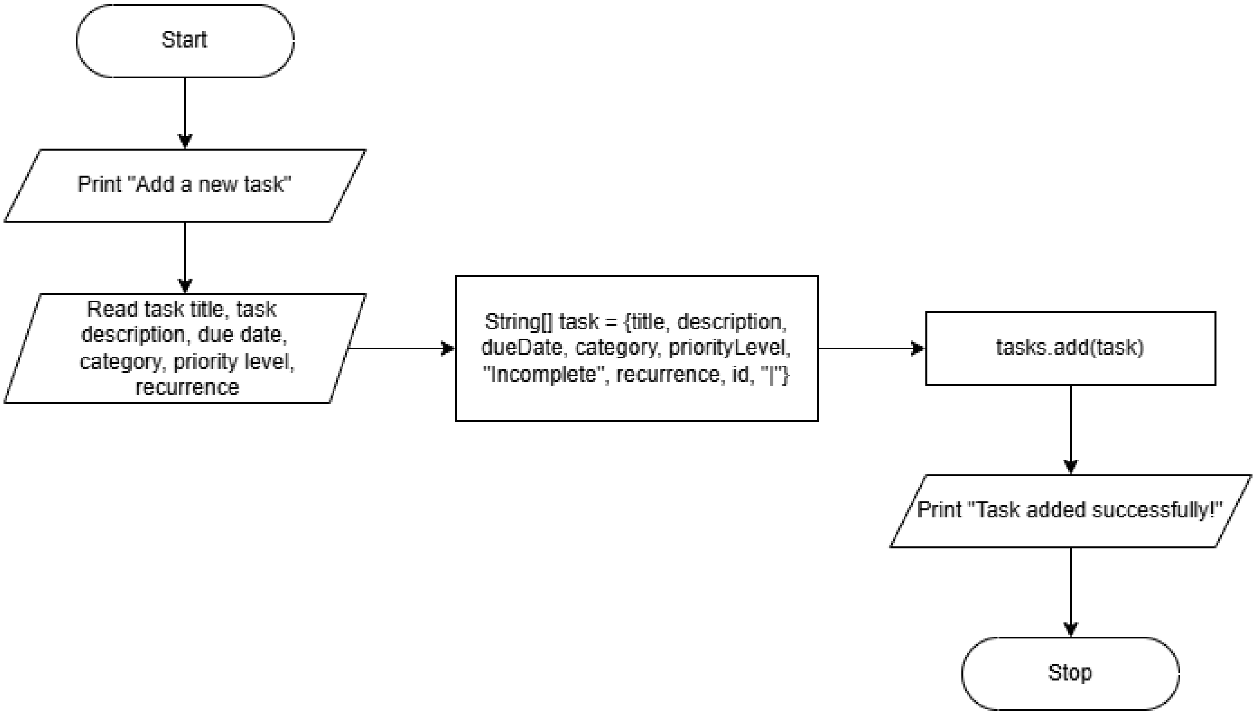
Attributes

- Title: The name of the task.
- Description: A brief explanation of the task's purpose.
- Due Date: The deadline for the task, stored as a LocalDate in YYYY-MM-DD format.
- Category: The classification of the task (e.g., Homework, Personal, Work).
- Priority: Priority level of the task (Low, Medium, High).
- Status: Indicates whether the task is complete or incomplete.

2. Task Creation

- **User Interaction**:
  - Prompts for task title, description, due date, category, and priority level.
  - Validates user inputs for the due date (ensures the format is YYYY-MM-DD) and priority level (Low, Medium, High). Invalid inputs trigger error messages and re-prompt the user.
- **Code Implementation**:

  addTask():
  - Reads user inputs via Scanner.
  - Creates a String[] array to store task attributes (e.g., title, description, due date).
  - Adds the task to the shared ArrayList<String[]>tasks
  - Default values: All new tasks are marked as "Incomplete".
- **Output**: Confirms task creation with a success message.


Start
Print "Add a new task"
Read task title, task description, due date, category, priority level, recurrence
String[] task = {title, description, dueDate, category, priorityLevel, "Incomplete", recurrence, id, "|"}
tasks.add(task)
Print "Task added successfully!"
Stop

3. Task Management

Purpose: View and manage existing tasks, including marking them as complete and handling recurring tasks.

- **User Interaction:**
    - Prompts for a task number to mark as complete.
- **Code Implementation:**

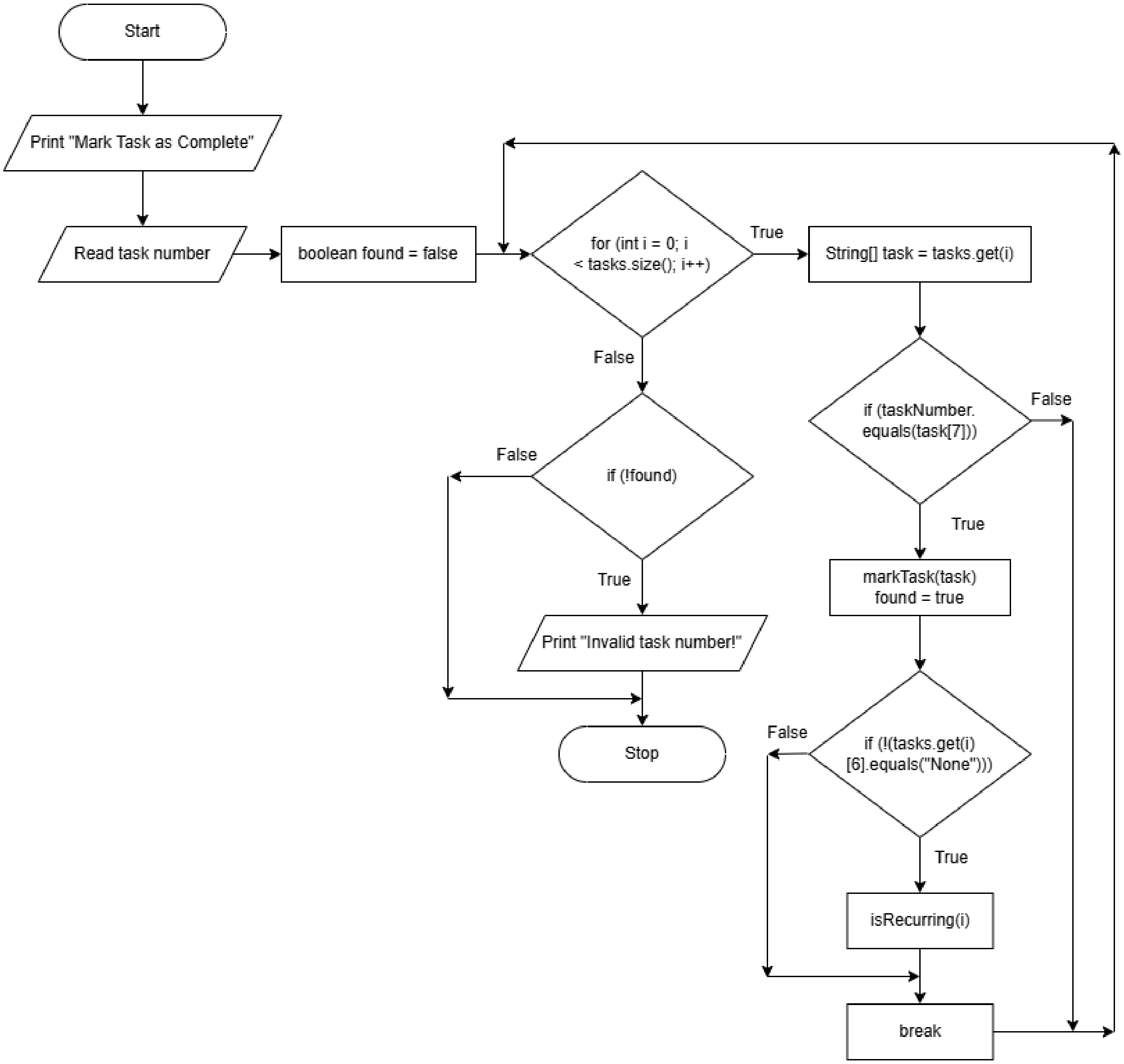
  markTaskAsComplete():
    - Validates user input to ensure the task number is within range.
    - Updates the task’s status to "Complete".
    - Checks if the task is recurring and, if so, triggers the creation of a new task with an updated due date using isRecurring().

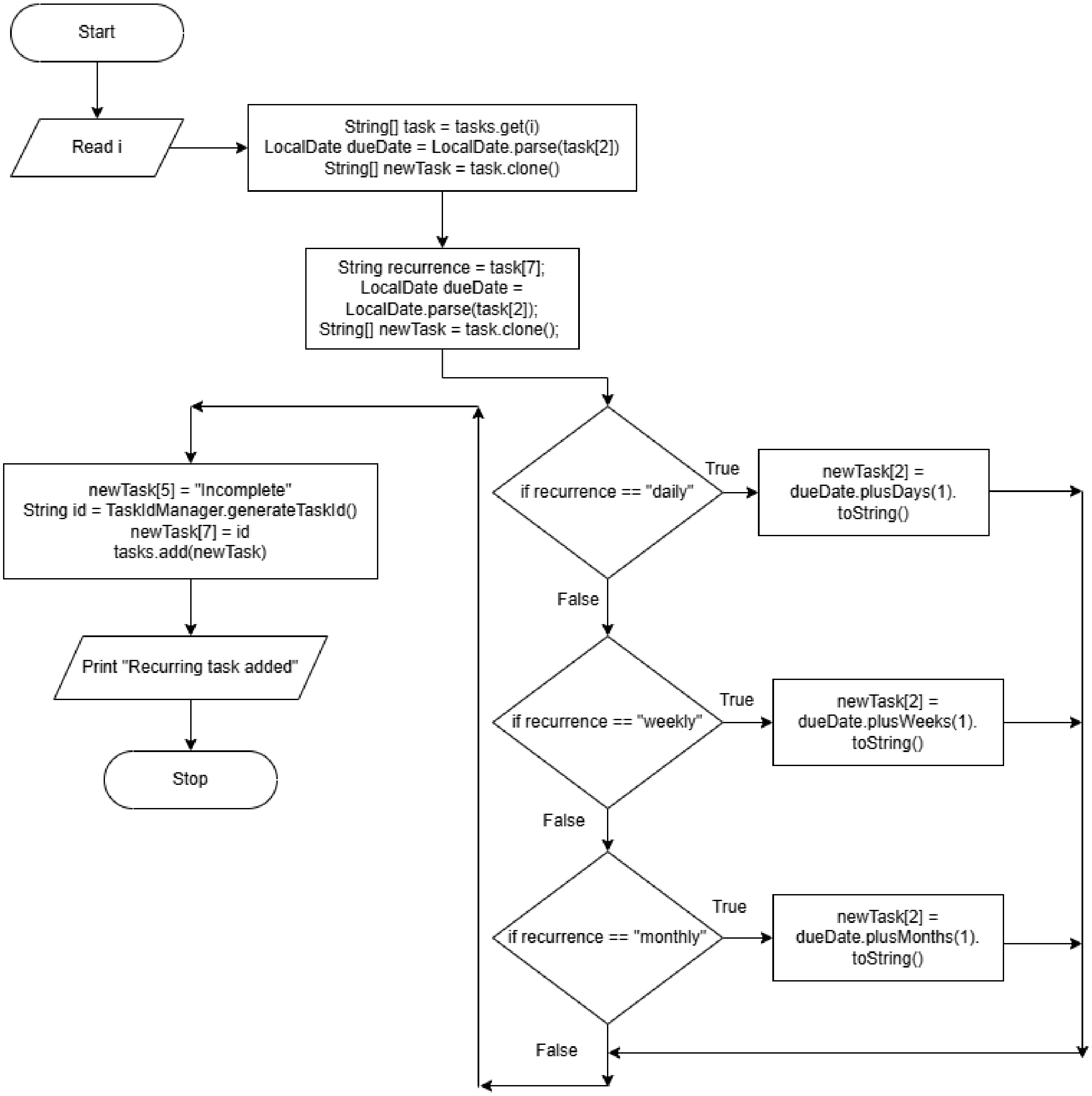
  isRecurring():

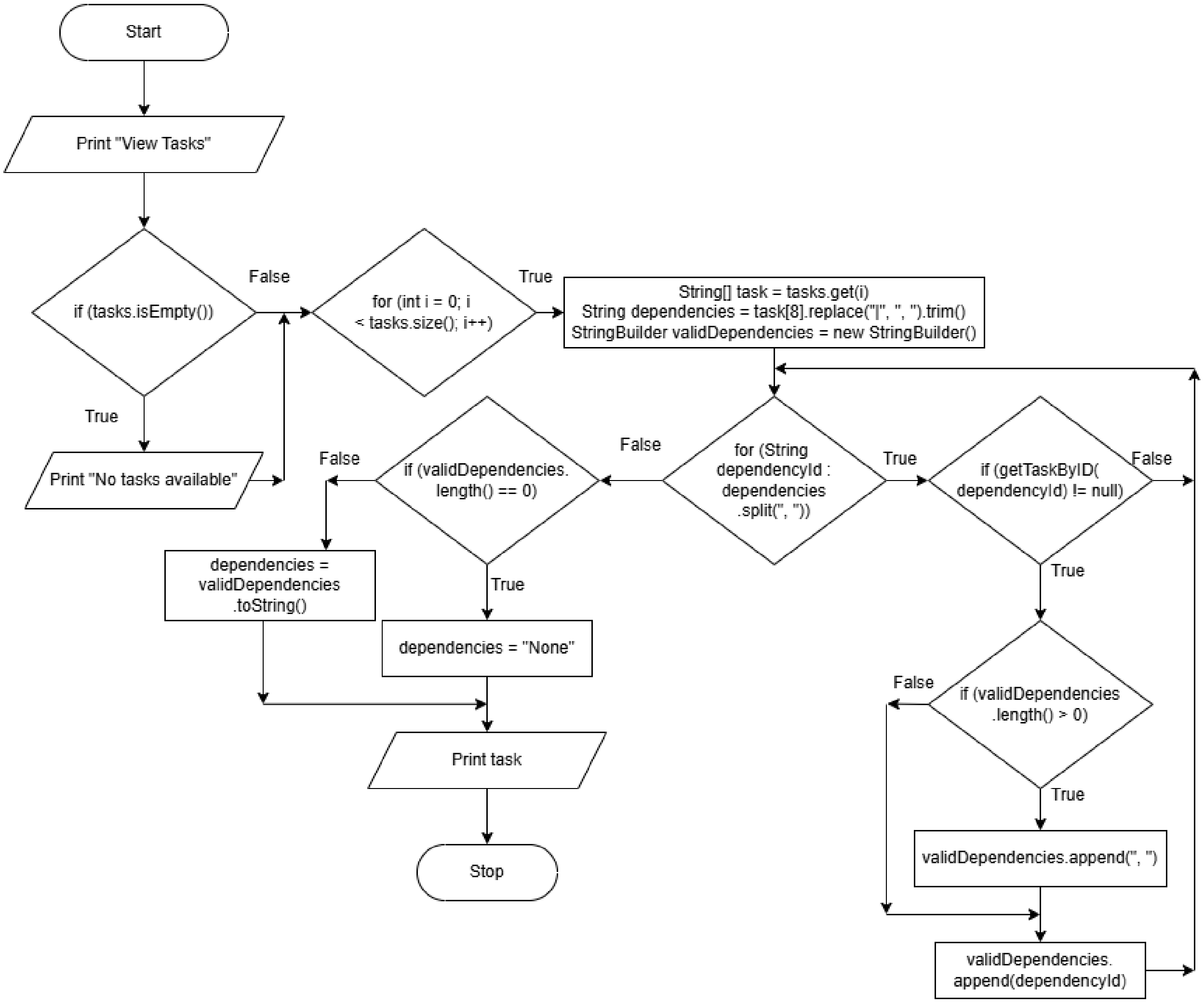
    - Generates a new task by cloning the original and adjusting the due date based on the recurrence interval (Daily, Weekly, Monthly).

  viewTasks();

    - To display all tasks in a structured format, including their details like task ID, description, due date, category, priority, and dependencies.
    - Checks if the tasks list is empty using tasks.isEmpty(). If true, it prints "No tasks available."
    - If there are tasks, it iterates through the list using a for loop
    - Provides a clean and readable output by formatting dependencies and ensuring missing dependencies are labeled as "None."
- Output:
    - Displays a success message indicating the task is marked complete.
    - For recurring tasks, inform the user about the newly created recurring task.
    - Display a list of tasks added by users in detail.


Start
Print "Mark Task as Complete"
Read task number
boolean found = false
for (int i = 0; i < tasks.size(); i++)
True
String[] task = tasks.get(i)
False
if (taskNumber.equals(task[7]))
False
True
markTask(task)
found = true
if (!found)
False
True
Print "Invalid task number!"
Stop
if (!(tasks.get(i)[6].equals("None")))
False
True
isRecurring(i)
break


Start
Read i
String[] task = tasks.get(i)
LocalDate dueDate = LocalDate.parse(task[2])
String[] newTask = task.clone()
String recurrence = task[7];
LocalDate dueDate =
LocalDate.parse(task[2]);
String[] newTask = task.clone();
if recurrence == "daily"
True
newTask[2] =
dueDate.plusDays(1).
toString()
False
if recurrence == "weekly"
True
newTask[2] =
dueDate.plusWeeks(1).
toString()
False
if recurrence == "monthly"
True
newTask[2] =
dueDate.plusMonths(1).
toString()
False
newTask[5] = "Incomplete"
String id = TaskIdManager.generateTaskId()
newTask[7] = id
tasks.add(newTask)
Print "Recurring task added"
Stop


Start
Print "View Tasks"
if (tasks.isEmpty())
False
True
Print "No tasks available"
for (int i = 0; i < tasks.size(); i++)
True
String[] task = tasks.get(i)
String dependencies = task[8].replace("|", ", ").trim()
StringBuilder validDependencies = new StringBuilder()
for (String dependencyId : dependencies .split(", "))
False
True
if (getTaskByID(dependencyId) != null)
False
True
if (validDependencies .length() > 0)
False
True
validDependencies.append(", ")
validDependencies. append(dependencyId)
if (validDependencies. length() == 0)
False
True
dependencies = validDependencies .toString()
dependencies = "None"
Print task
Stop

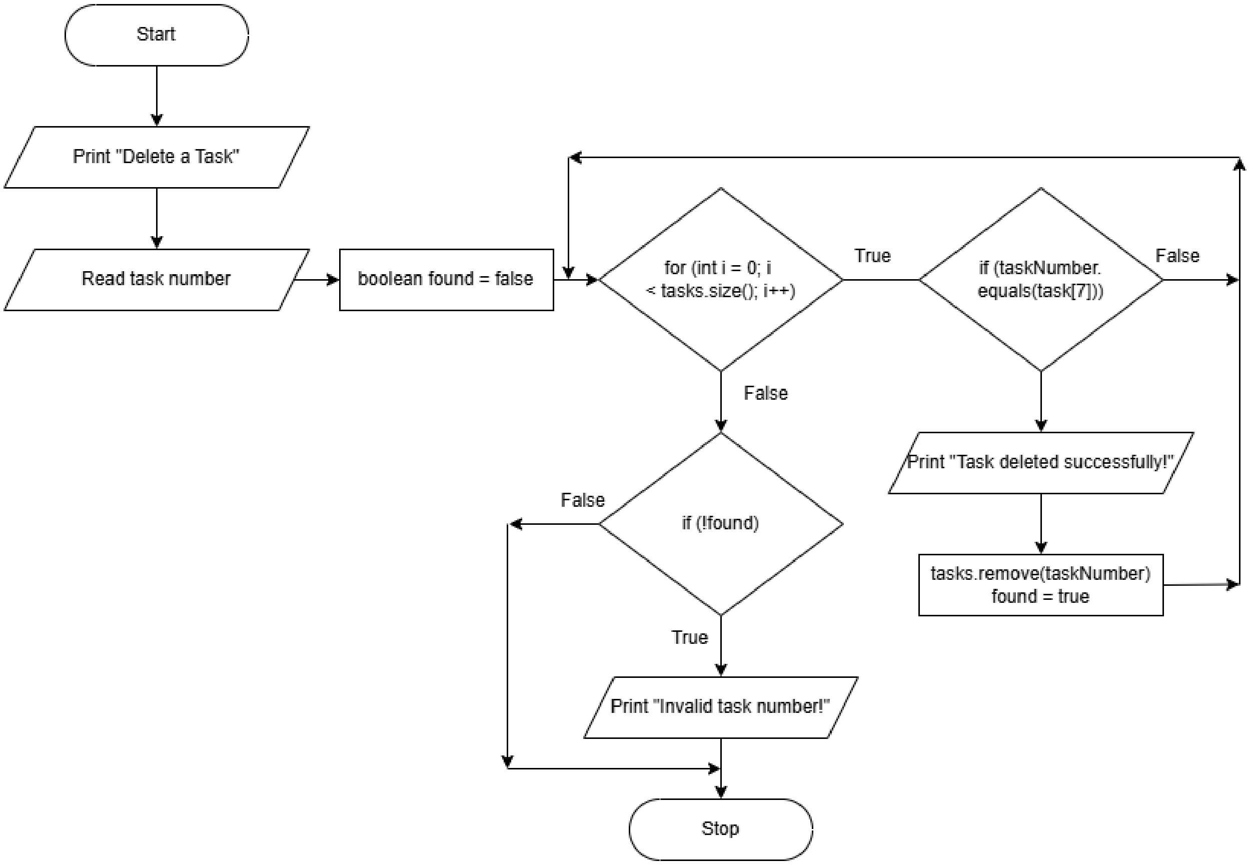
4. Task Deletion

Purpose: Permanently removes a selected task

- **User Interaction:**
  - Prompts the user for a task number to delete.
- **Code Implementation:**

  deleteTask():
  - Validates the input task number.
  - Removes the corresponding task from the tasks list using ArrayList.remove().
- **Output:**
  - Displays a confirmation message indicating the task has been deleted.

- Handles invalid task numbers with an error message.


Start
Print "Delete a Task"
Read task number
boolean found = false
for (int i = 0; i < tasks.size(); i++)
True
if (taskNumber. equals(task[7]))
False
False
Print "Task deleted successfully!"
tasks.remove(taskNumber) found = true
False
if (!found)
True
Print "Invalid task number!"
Stop

5. Task Sorting

Purpose: Organizes tasks for better viewing, based on due dates or priority levels.

- **User Interaction:**
    - Prompts for a sorting preference:
    - Due Date (Ascending)
    - Due Date (Descending)
    - Priority (High to Low)
    - Priority (Low to High).
- **Code Implementation:**

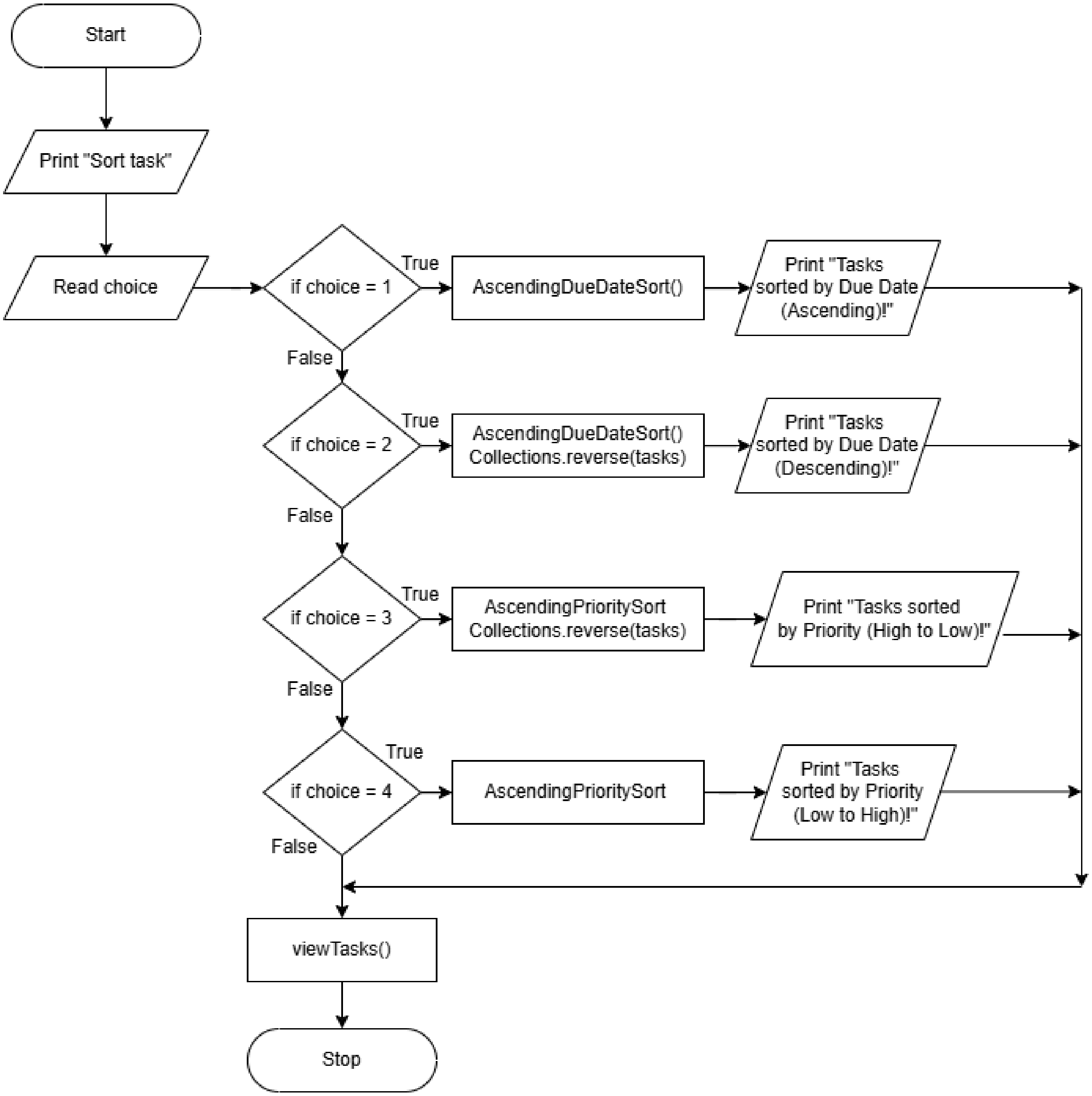
  sortTasks():

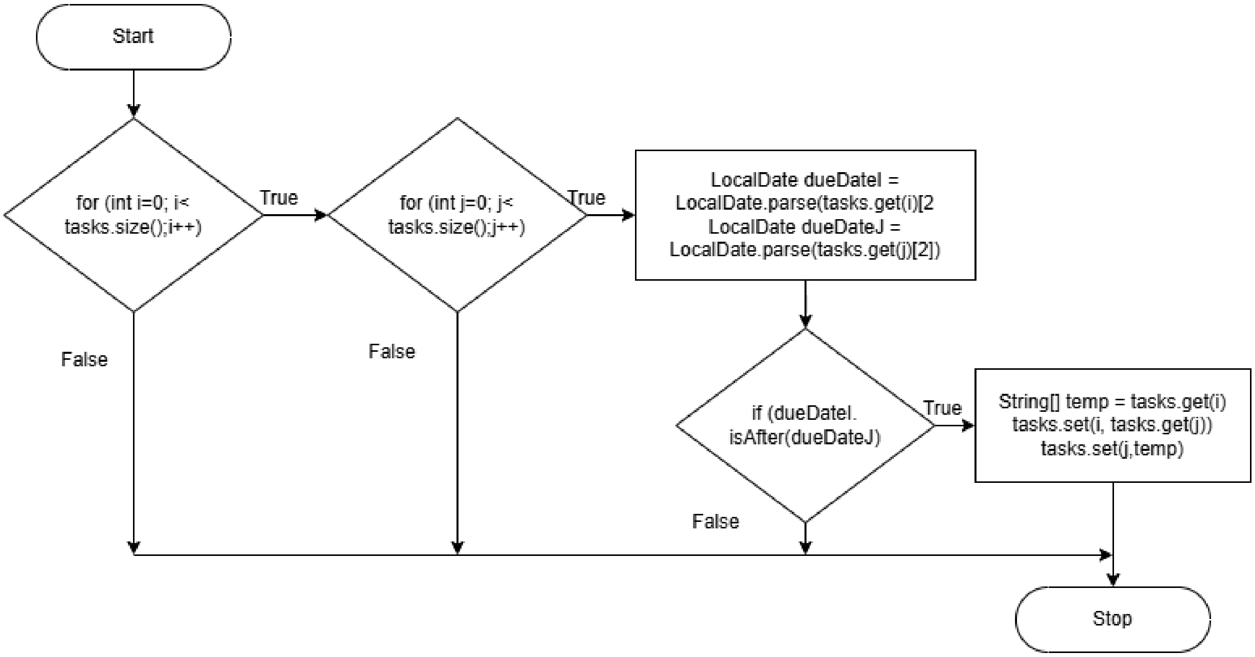
  Handles the user prompt and delegates sorting to specific helper methods:

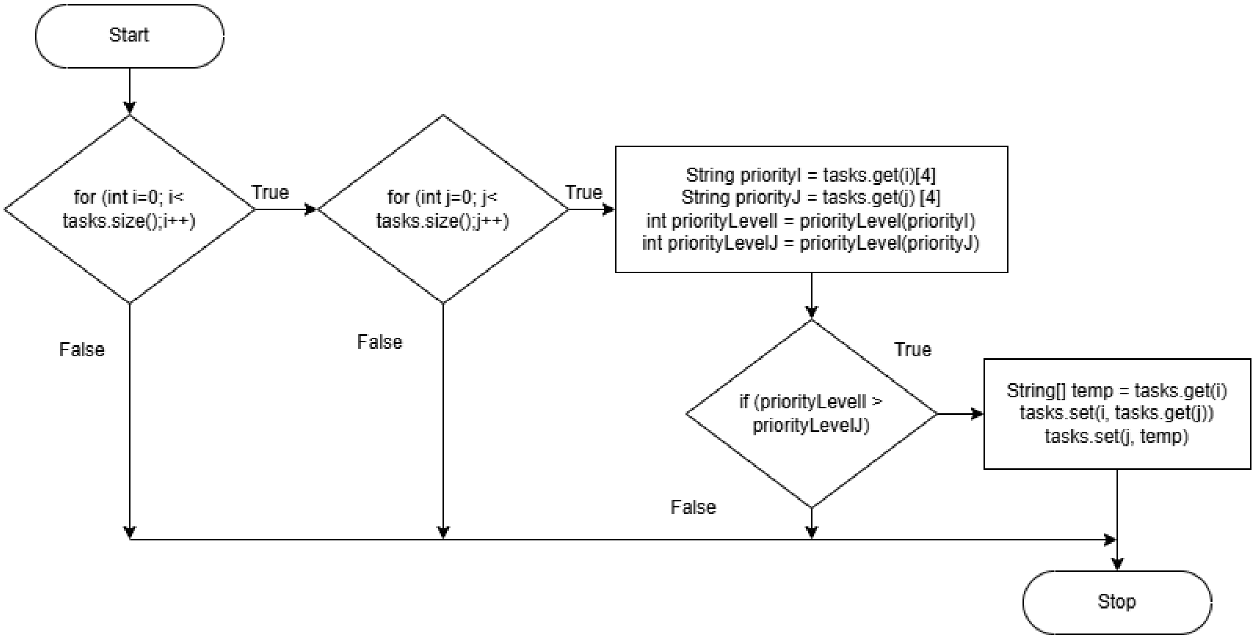
    - AscendingDueDateSort(): Compares tasks by their due date (LocalDate.parse() for comparison).

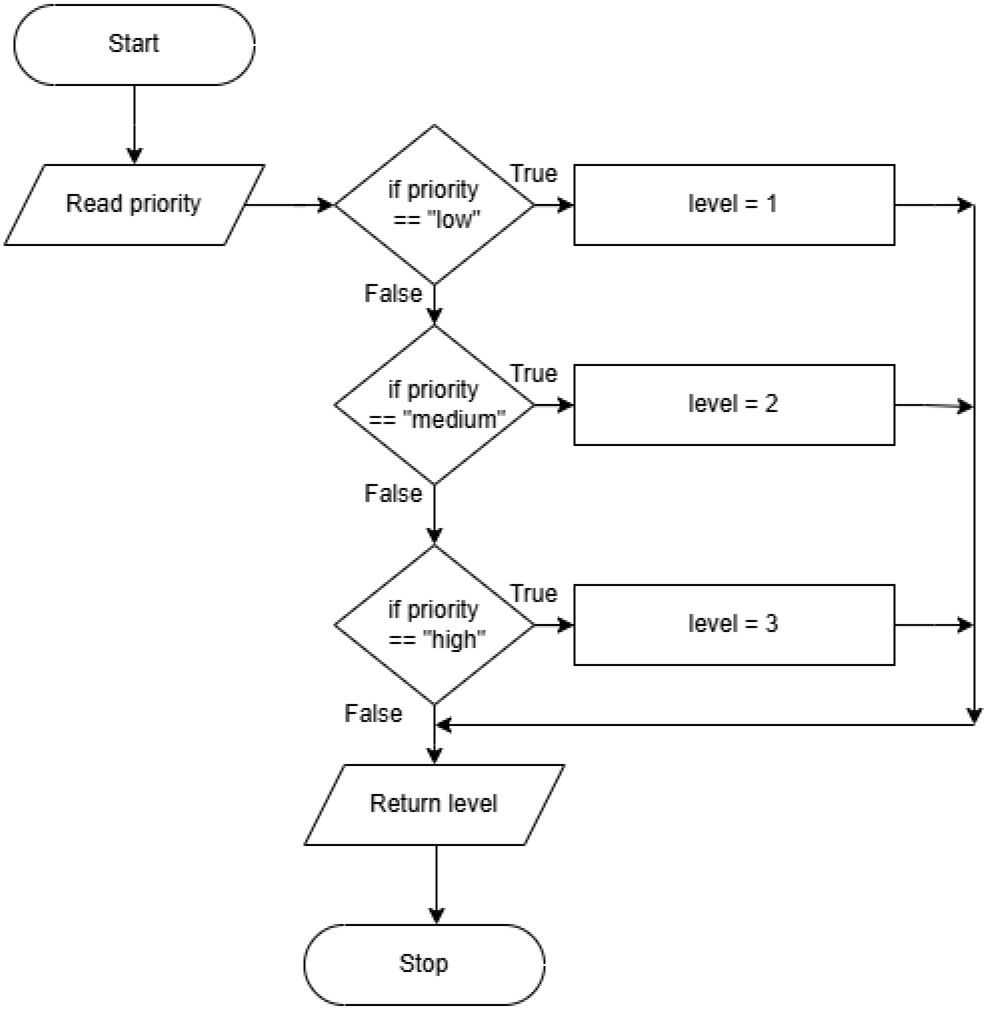
- AscendingPrioritySort(): Converts priority levels into integers for sorting.
- Sorts the tasks list and reverses it for descending order where needed.

- Output:
  - Displays tasks in the sorted order using viewTasks().
  - Notifies the user of the chosen sorting method


Start
Print "Sort task"
Read choice
if choice = 1
True
False
AscendingDueDateSort()
Print "Tasks sorted by Due Date (Ascending)!"
if choice = 2
True
False
AscendingDueDateSort() Collections.reverse(tasks)
Print "Tasks sorted by Due Date (Descending)!"
if choice = 3
True
False
AscendingPrioritySort Collections.reverse(tasks)
Print "Tasks sorted by Priority (High to Low)!"
if choice = 4
True
False
AscendingPrioritySort
Print "Tasks sorted by Priority (Low to High)!"
viewTasks()
Stop


Start
for (int i=0; i< tasks.size();i++)
True
for (int j=0; j< tasks.size();j++)
True
LocalDate dueDateI = LocalDate.parse(tasks.get(i)[2
LocalDate dueDateJ = LocalDate.parse(tasks.get(j)[2])
False
False
if (dueDateI. isAfter(dueDateJ)
True
String[] temp = tasks.get(i)
tasks.set(i, tasks.get(j))
tasks.set(j,temp)
False
Stop


Start
for (int i=0; i< tasks.size();i++)
True
for (int j=0; j< tasks.size();j++)
True
String priorityI = tasks.get(i)[4]
String priorityJ = tasks.get(j) [4]
int priorityLevelI = priorityLevel(priorityI)
int priorityLevelJ = priorityLevel(priorityJ)
False
False
True
if (priorityLevelI > priorityLevelJ)
String[] temp = tasks.get(i)
tasks.set(i, tasks.get(j))
tasks.set(j, temp)
False
Stop

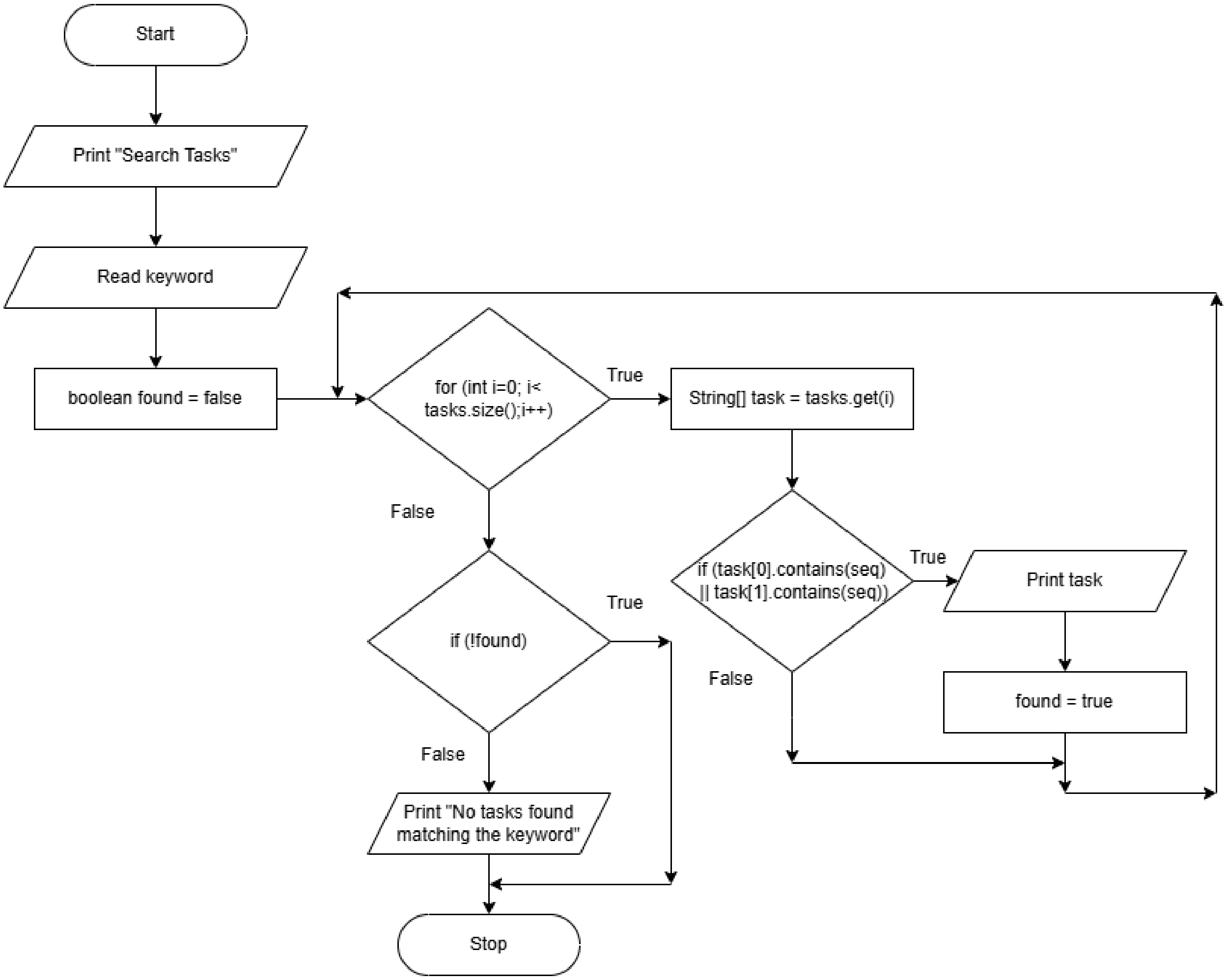


6. Task Searching

Purpose: Quickly locates tasks by keyword in the title or description.

- **User Interaction:**
    - Prompts the user to enter a keyword for the search.
- **Code Implementation:**

  searchTasks():
    - Iterates over all tasks, checking if the title or description contains the keyword.
    - Uses String.contains() for case-sensitive matching.
- **Output:**
    - Lists matching tasks with their details, including title, due date, and priority.
    - If no matches are found, inform the user with a "No tasks found" message.


Start
Print "Search Tasks"
Read keyword
boolean found = false
for (int i=0; i< tasks.size();i++)
True
String[] task = tasks.get(i)
False
if (task[0].contains(seq) || task[1].contains(seq))
True
Print task
False
found = true
if (!found)
True
False
Print "No tasks found matching the keyword"
Stop

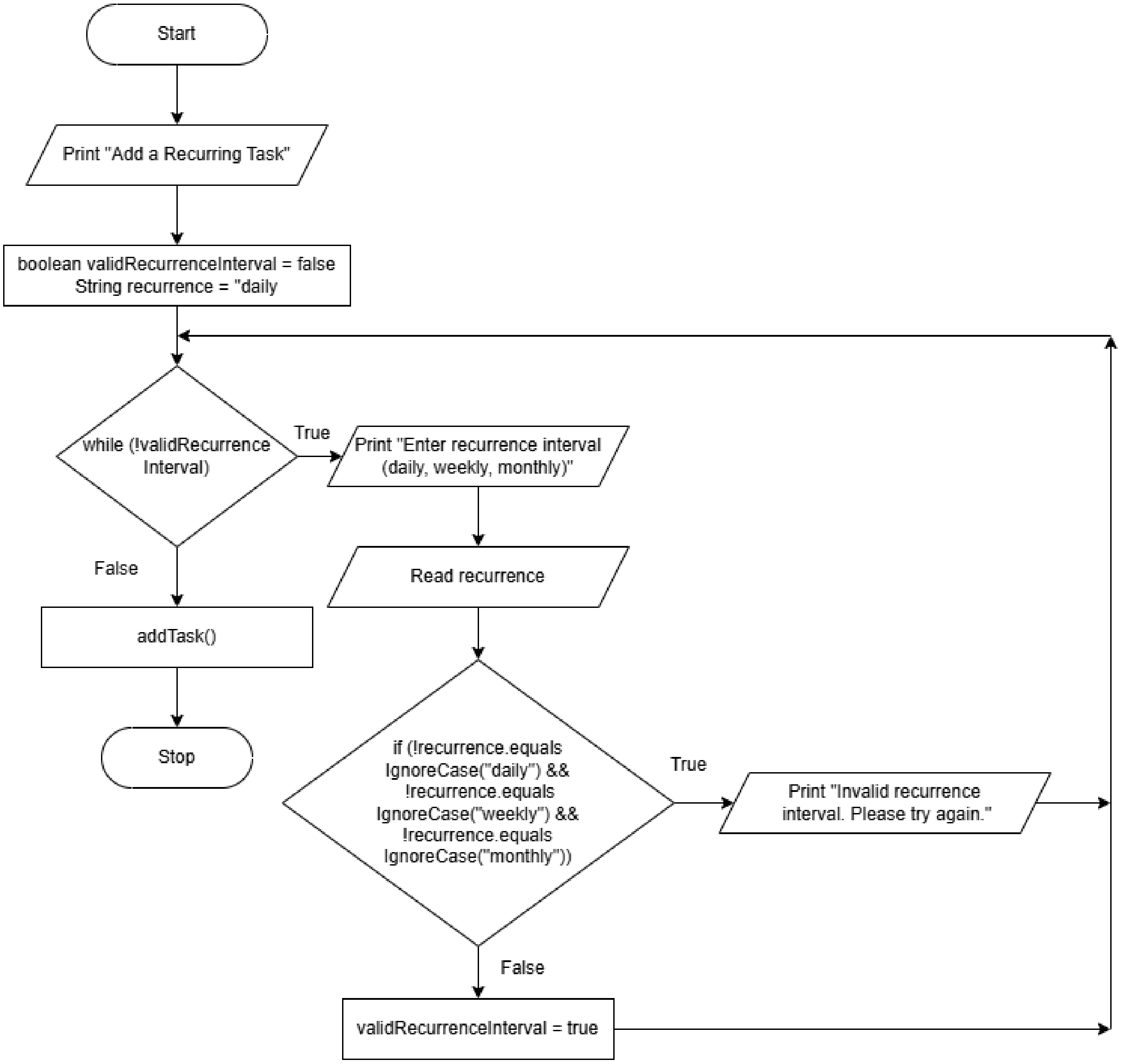
7. Task Recurring

Purpose: Adds recurring tasks and manages their regeneration upon completion.

- **User Interaction:**
  - Prompts for task details (title, description, due date, category, priority level, and recurrence interval).
  - Validates the recurrence interval input (Daily, Weekly, Monthly).
- **Code Implementation:**

  ReccuringTask():
  - Similar to addTask(), but includes an additional attribute for recurrence interval.
  - Sets the recurring flag (task[6] = "true") and saves the recurrence type (task[7]).
- **Output:**

- Confirms the creation of a recurring task.
- On completion, generates a new task using the isRecurring() method, updating the due date appropriately.


Start
Print "Add a Recurring Task"
boolean validRecurrenceInterval = false
String recurrence = "daily
while (!validRecurrence Interval)
True
Print "Enter recurrence interval (daily, weekly, monthly)"
False
Read recurrence
addTask()
Stop
if (!recurrence.equals IgnoreCase("daily") && !recurrence.equals IgnoreCase("weekly") && !recurrence.equals IgnoreCase("monthly"))
True
Print "Invalid recurrence interval. Please try again."
False
validRecurrenceInterval = true

8. Task Dependencies

Purpose:

- Set task dependencies: Allow users to define dependencies between tasks, ensuring that one task cannot be completed until its dependent tasks are completed.
- Check for cyclic dependencies: Prevent circular dependencies that could create infinite loops or logical errors.
- Mark tasks as complete: Ensure tasks are only marked as complete if all their dependencies are already completed.

- **User Interaction:**

  Set Task Dependencies:

  - The user is prompted to enter the IDs of two tasks:
    1. The dependent task (the task that depends on another).
    2. The preceding task (the task that must be completed first).
  - The system checks for:
    1. Invalid task IDs.
    2. Self-dependencies (a task cannot depend on itself).
    3. Cyclic dependencies (a task cannot depend on another task that indirectly depends on it).
    4. If all checks pass, the dependency is added.

  Mark Task as Complete:

  - The system checks if all dependencies of the task are completed.
  - If any dependency is incomplete, a warning is displayed.
  - If all dependencies are complete, the task is marked as complete.

- **Code Implementation:**

  taskDependencies():

  - Prompts the user to input dependent and preceding task IDs.
  - Validates the inputs and checks for self-dependencies and cyclic dependencies.
  - Updates the task's dependency list if valid.

isCyclic() & hasCycle():

- Recursively checks if adding a dependency would create a cycle in the task dependency graph.

getTaskByID():

- Retrieves a task from the tasks list using its ID.

markTask():

- Checks if all dependencies of a task are complete.
- Marks the task as complete if all dependencies are satisfied; otherwise, displays a warning.

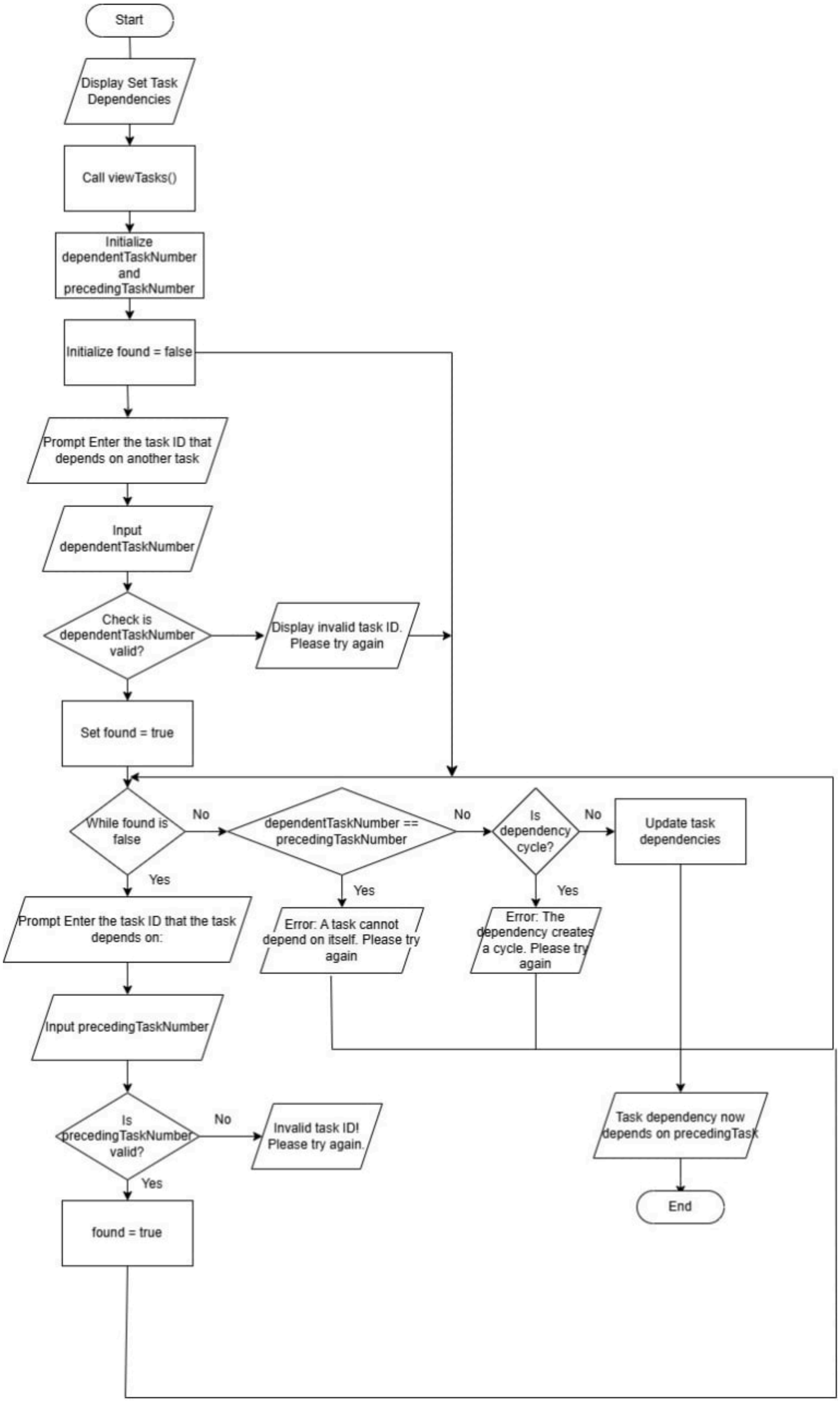
Output:

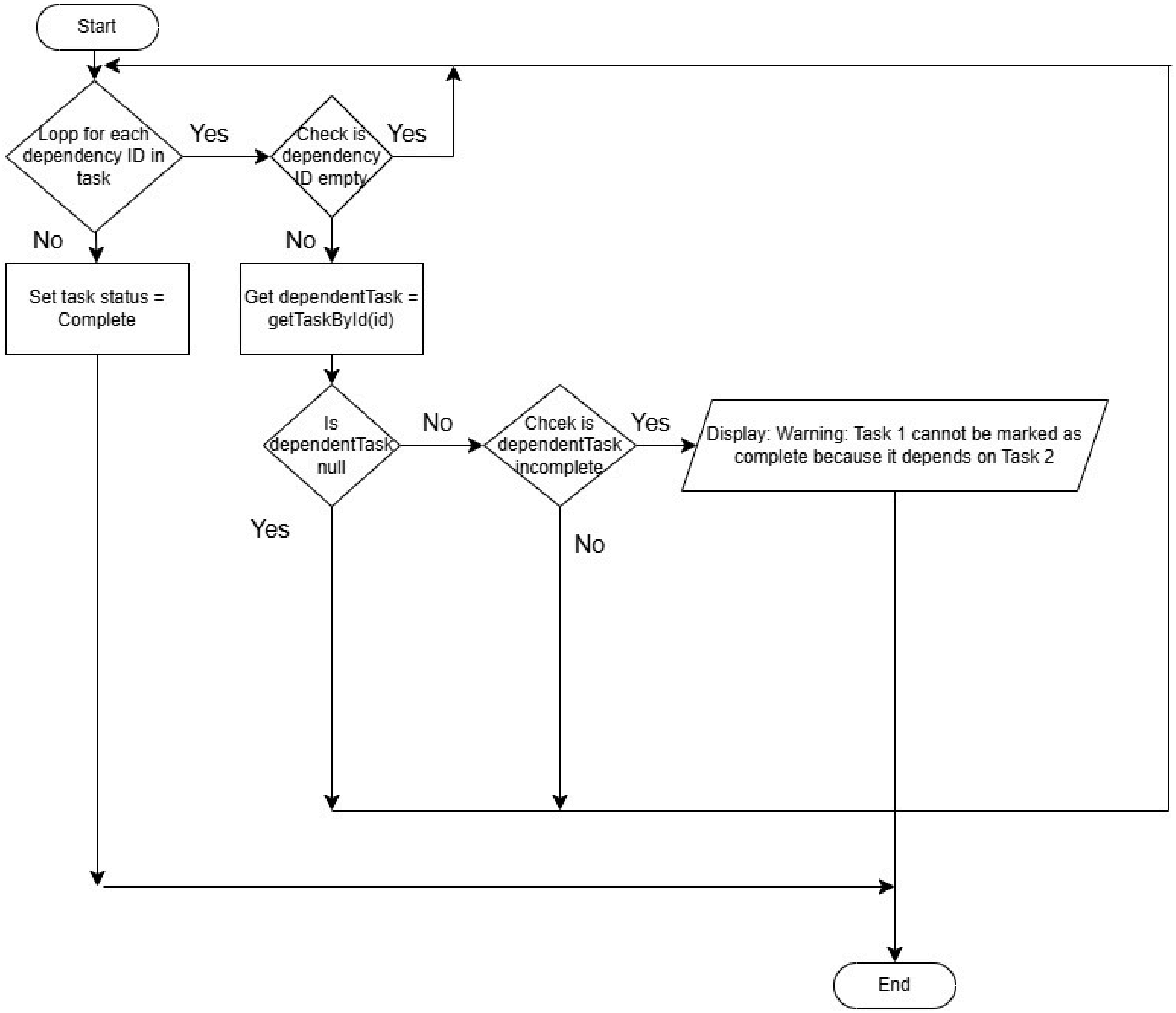
Set Task dependencies:

- If successful: Task "Task A" now depends on "Task B".
- If invalid: Invalid task ID! Please try again.
- If cyclic: Error: This dependency creates a cycle. Please try again.
- If self-dependency: Error: A task cannot depend on itself. Please try again.

Mark Task as Complete:

- If successful: Task "Task A" marked as complete!
- If dependencies are incomplete: Warning: Task "Task A" cannot be marked as complete because it depends on "Task B". Please complete the dependent task first.


Start
Display Set Task Dependencies
Call viewTasks()
Initialize dependentTaskNumber and precedingTaskNumber
Initialize found = false
Prompt Enter the task ID that depends on another task
Input dependentTaskNumber
Check is dependentTaskNumber valid?
Display invalid task ID. Please try again
Set found = true
While found is false
No
Yes
dependentTaskNumber == precedingTaskNumber
No
Yes
Is dependency cycle?
No
Yes
Update task dependencies
Prompt Enter the task ID that the task depends on:
Error: A task cannot depend on itself. Please try again
Error: The dependency creates a cycle. Please try again
Input precedingTaskNumber
Is precedingTaskNumber valid?
No
Invalid task ID! Please try again.
Yes
Task dependency now depends on precedingTask
End
found = true


Start
Lopp for each dependency ID in task
Yes
Check is dependency ID empty
Yes
No
No
Set task status = Complete
Get dependentTask = getTaskById(id)
Is dependentTask null
No
Chcek is dependentTask incomplete
Yes
Display: Warning: Task 1 cannot be marked as complete because it depends on Task 2
Yes
No
End

9. Edit Task

Purpose: to allow the user to edit an existing task in a task management system. The user can modify specific attributes of a task, such as its title, description, due date, category, priority, or dependencies.

User Interaction:

- The user is shown a list of tasks (viewTasks()).
- The user is prompted to select a task number to edit.
- The user is shown a menu of options to edit:

    - ➢ Title
    - ➢ Description
    - ➢ Due Date
    - ➢ Category
    - ➢ Priority
    - ➢ Set Task Dependency
    - ➢ Cancel
- The user selects an option and provides the necessary input (e.g., new title, new due date, etc.).
- The system updates the task and displays the updated list of tasks.

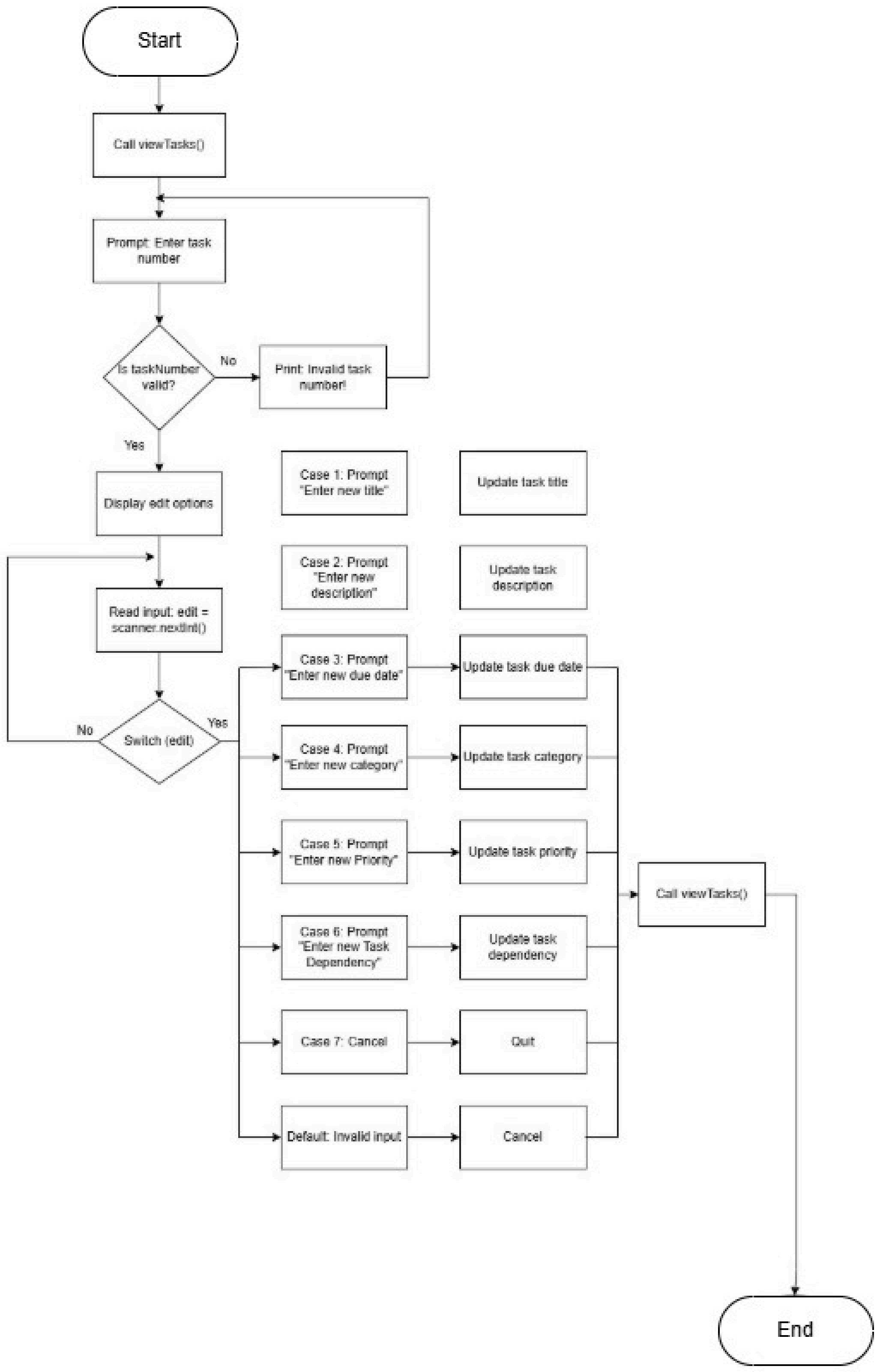
Code Implementation:

- Display Tasks: The viewTasks() method is called to show the current list of tasks.
- Input Task Number: The user is prompted to enter the task number to edit. The input is validated to ensure it is within the valid range.
- Edit Options: A menu is displayed, allowing the user to choose what to edit.
- Switch Case: Based on the user's choice, the corresponding task attribute is updated:
    - ➢ Title: Update the task title.
    - ➢ Description: Update the task description.
    - ➢ Due Date: Validate and update the due date.
    - ➢ Category: Update the task category.
    - ➢ Priority: Validate and update the priority level.
    - ➢ Task Dependency: Call taskDependencies() to set dependencies.
    - ➢ Cancel: Return to the main menu.
- Output Updated Tasks: After editing, the updated list of tasks is displayed using viewTasks().

Output:

- Initial Output:
    - ➢ Displays the list of tasks.
    - ➢ Prompts the user to enter a task number.

- Edit Output:
    - ➢ Displays the selected task's current details.
    - ➢ Prompts the user to enter new values for the selected attribute.
- Final Output:
    - ➢ Confirms the update (e.g., "Task 'Old Title' has been updated to 'New Title'.").
    - ➢ Displays the updated list of tasks.
- Error Output:
    - ➢ If the task number is invalid: "Invalid task number!"
    - ➢ If the input is invalid (e.g., wrong date format or priority level): "Invalid input. Please try again."


Start
Call viewTasks()
Prompt: Enter task number
Is taskNumber valid?
No
Print: Invalid task number!
Yes
Display edit options
Read input: edit = scanner.nextInt()
Switch (edit)
No
Yes
Case 1: Prompt "Enter new title"
Update task title
Case 2: Prompt "Enter new description"
Update task description
Case 3: Prompt "Enter new due date"
Update task due date
Case 4: Prompt "Enter new category"
Update task category
Case 5: Prompt "Enter new Priority"
Update task priority
Case 6: Prompt "Enter new Task Dependency"
Update task dependency
Case 7: Cancel
Quit
Default: Invalid input
Cancel
Call viewTasks()
End

Extra Features

1. **Graphical User Interface (GUI)**

   Solution: JavaFX

   Purpose: The GUI provides an intuitive and user-friendly interface for managing tasks. Users can add, edit, delete and mark tasks as complete or incomplete. The application also allows sorting tasks by title and saves tasks to a CSV file for persistence, ensuring that tasks are retained even after the application is closed. This application is designed to help users organise their tasks efficiently and keep track of their progress.

   User Interaction:

   a) **Main Interface**

   - **Header:** Displays the title "To-Do-List"

   - **Task List:** Displays all tasks in a scrollable list. Each task is shown with:
     - A checkbox to mark it as complete or incomplete
     - A "Delete" button to remove the task
     - An "Edit" button to modify the task details

   - **Add Task Section:** Allows the user to input
     - Task Title
     - Task Description
     - Due Date (using a date picker)
     - Category (eg. Work, Personal, Homework)
     - Priority (eg. Low, Medium, High)

- **Sort Buttons:** Two buttons to sort tasks alphabetically by title in ascending or descending order.

b) User Action

- Add a task:
  - Fill in the task details in the input fields
  - Click the “Add Task” button to add the task to the list

- Mark a Task as Complete/Incomplete
  - Click the checkbox next to a task to toggle its status

- Edit a Task:
  - Click the “Edit” button next to a task to open a dialog box
  - Modify the task details and click “Save” to update the task

- Delete a Task:
  - Click the “Delete” button next to a task to remove it from the list

- Sort Tasks:
  - Click the “Sort Ascending” or “Sort Descending” button to sort the tasks alphabetically by title

Code Implementation:

a) **Main class**

- **start (Stage primaryStage):** initialize the GUI and sets up the main layout
- **createTaskListView():** Creates the task list view with checkboxes, delete and edit buttons
- **createAddTaskBox():** Creates the input fields and button for adding new tasks
- **createSortButtons():** Creates buttons for sorting tasks
- **loadTasksFromCSV():** Loads tasks from a CSV file into the task list
- **saveTasksToCSV():** Saves the current task list to a CSV file
- **showAlert(String title, String message):** Displays an alert dialog with a message
- **editTask(Task task):** Opens a dialog to edit the details of a selected task

b) **Task class**

- **title:** The name of the task
- **description:** A brief description of the task
- **dueDate:** The deadline for the task
- **category:** The category of the task (eg. Work, personal, Homework)
- **priority:** The priority level of the task (eg. Low, Medium, High)
- **status:** The current status of the task (Complete/Incomplete)

c) **TaskStorage class**

- **loadTasksFromCSV():** Reads tasks from a CSV file and returns them as a list
- **saveTaskToCSV(ObservableList<Task> tasks):** Writes the current task list to a CSV file

Output:

a) **Application window:**

- Displays a header titled "To-Do-List"
- Shows a form for adding new tasks with fields for title, description, due date, category and priority
- Displays a list of tasks with checkboxes, edit buttons and delete buttons
- Provides "Sort Ascending" and "Sort Descending" buttons below the task list

b) **Task List:**

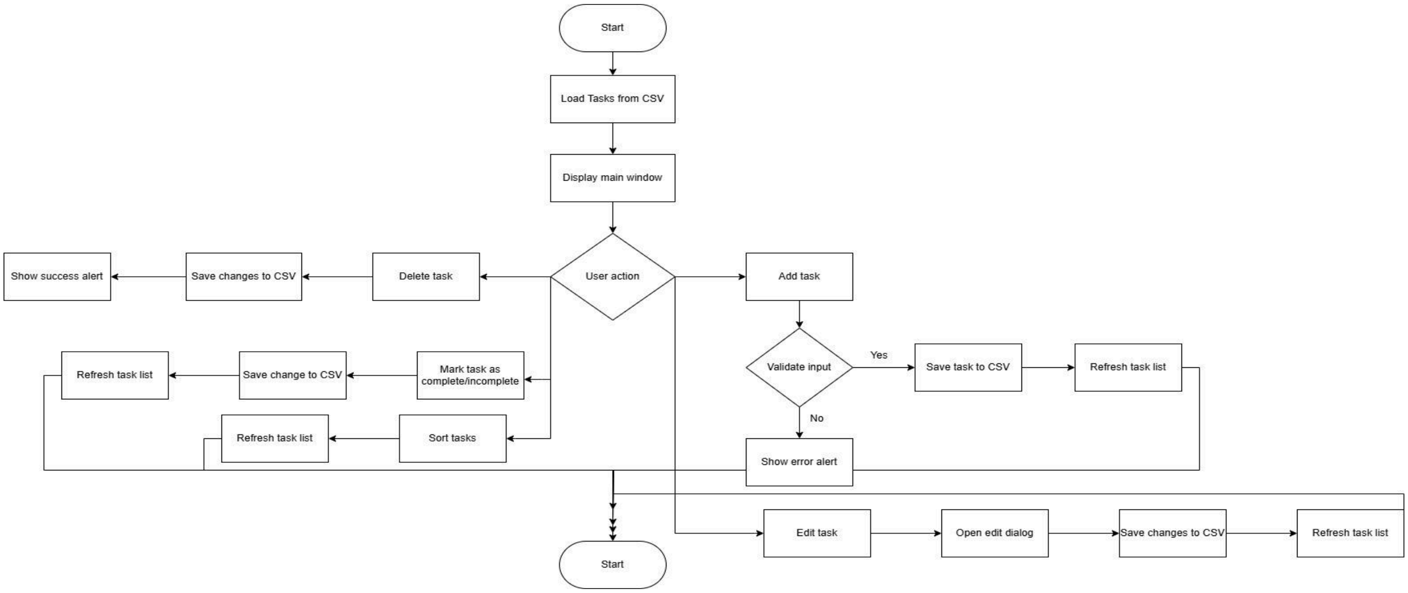
- Each task is displayed with its title, due date, priority and status
- Tasks marked as complete are checked, while incomplete tasks are unchecked

c) **Alerts:**

- Displays success or error messages for actions like deleting a task or failing to load/save tasks

d) **Edit dialog:**

- Opens a pop-up window with fields to edit the selected task's details



2. Email Notification System

Solution: JavaMail

Purpose: Email integration is to send automated email reminders to users about their upcoming tasks. This ensures that users are notified of tasks due within the next 24 hours.

User Interaction:

a) Input:

- The program loads tasks from a CSV file
- The user is prompted to enter their email address to receive the reminder

b) Output:

- If a task is due within 24 hours and is incomplete, the program sends an email reminder to the user
- If no tasks are due within 24 hours, the program notifies the user and exits

Code Implementation:

a) Main method

- Loads tasks from a CSV file using **TaskStorage.loadTasksFromCSV()**.
- Iterates through the tasks and checks if any task is due within 24 hours and is complete
- Prompts the user to enter their email address
- Calls the **sendEmailReminder()** method to send the email

b) sendEmailReminder() method

- Configures the SMTP server settings for Gmail
- Creates a **MimeMessage** object to compose the email
- Sets the sender, recipient, subject and email body
- Uses HTML formatting to create a visually appealing email

- Sends the email using the **Transport.send()** method

c) Email format

- A header with a task reminder
- A list of task details (title, description, due date, category, priority)
- A footer with a disclaimer

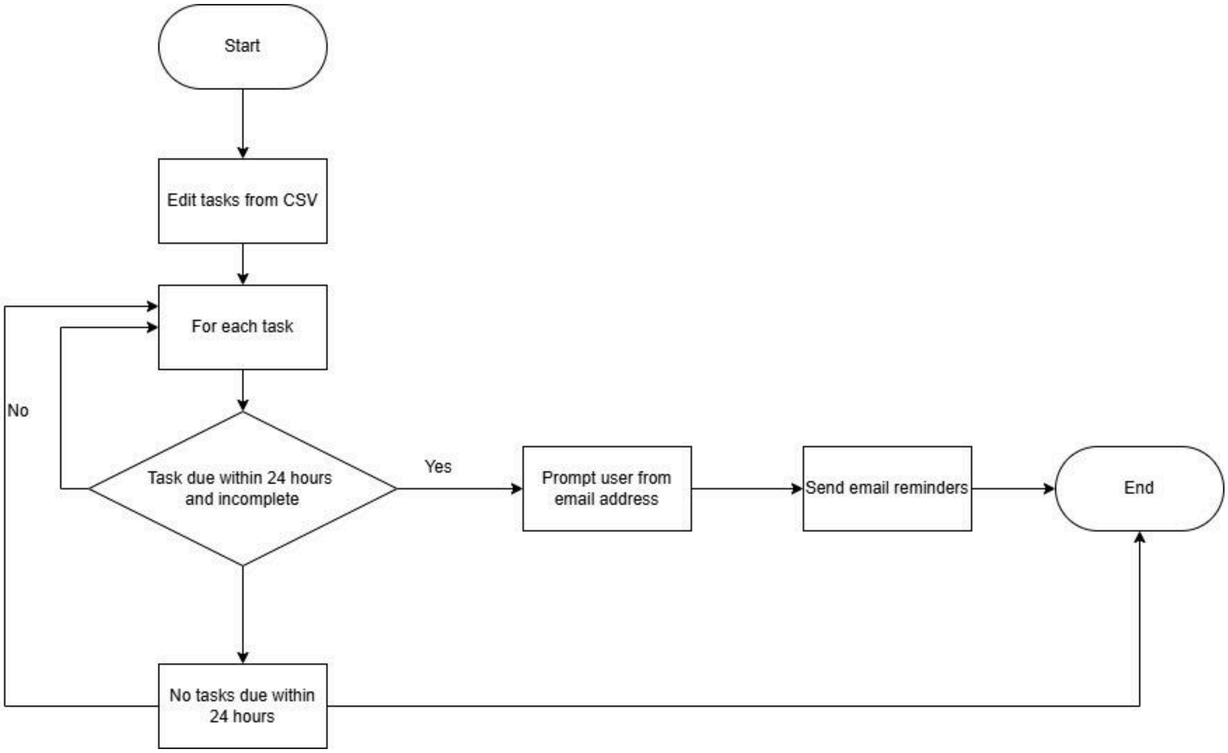
Output:

- The email subject is: 🔔 Task Reminder: [Task Title]
- The email body contains:
  - A greeting “Dear User,”
  - A reminder message: “This is a reminder that your task [Task title] is due on [Due Date].
  - Task details in a bulleted list:
    - ➔ Title
    - ➔ Description
    - ➔ Due date
    - ➔ Category
    - ➔ Priority
  - A closing message: “Please ensure that you complete it on time.”
  - A footer: “This is an automated email. Please do not reply.”
- If a reminder is sent:

```
Enter your email address to receive a reminder: user@example.com
Email reminder sent successfully to user@example.com
```

- If no tasks are due within 24 hours:

```
No tasks due within 24 hours
```


Start
Edit tasks from CSV
For each task
No
Task due within 24 hours and incomplete
Yes
Prompt user from email address
Send email reminders
End
No tasks due within 24 hours

3. Data Analytics

- Purpose : To provide a quick overview of users' task progress through an analytics dashboard that displays task counts, completion rates, and categorized summaries.

  Components :

  - **Total Tasks** – Shows the total number of tasks.
  - **Completed Tasks** – Shows how many tasks are finished.
  - **Pending Tasks** – Shows how many tasks are not done.
  - **Completion Rate** – Shows the percentage of tasks that are finished.

- **Task Categories** – Shows tasks grouped by types like Homework, Personal, and Work

Code Implementation

main(String[] args):

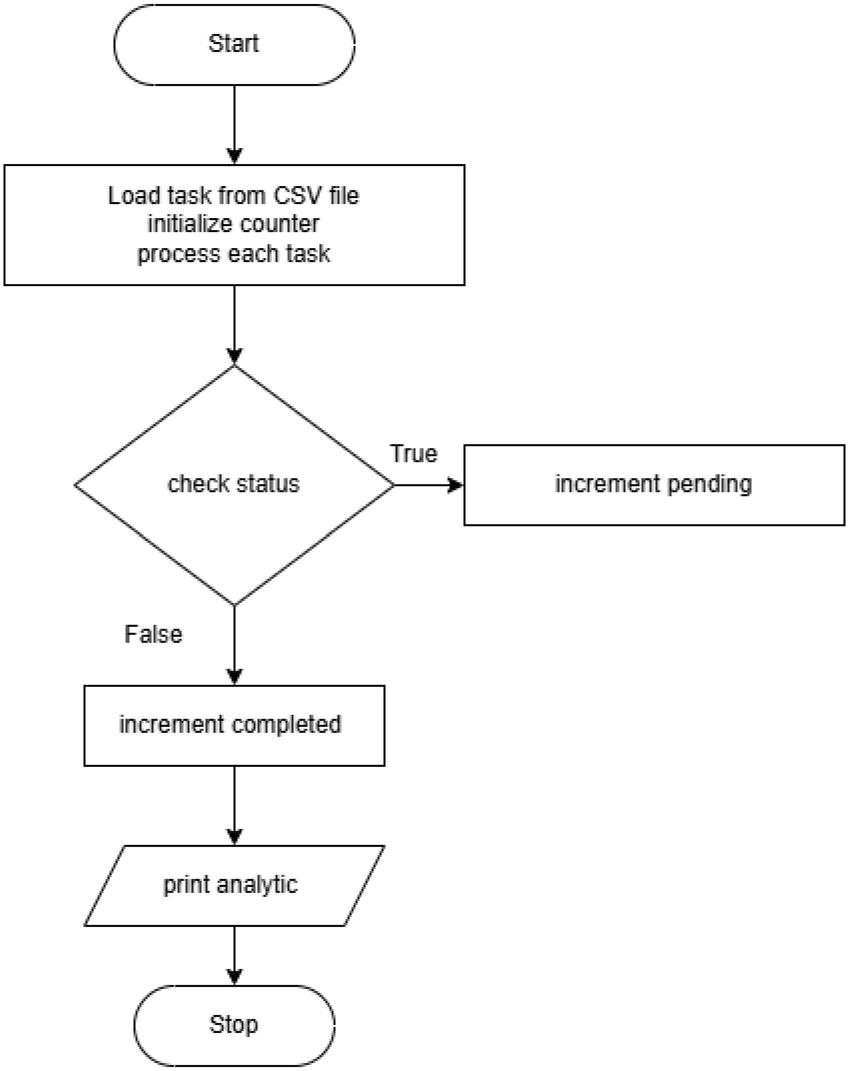
- This is the main method that runs the program.
- Loads tasks from a CSV file using TaskStorage.loadTasksFromCSV().
- Calls the showAnalytics() method to display the analytics dashboard.

showAnalytics(List<Task> tasks):

- This method processes the list of tasks to:
- Count total, completed, and pending tasks.
- Count tasks by category (Homework, Personal, Work).
- Calculate the completion rate.
- Display the analytics dashboard in the console.

Output:

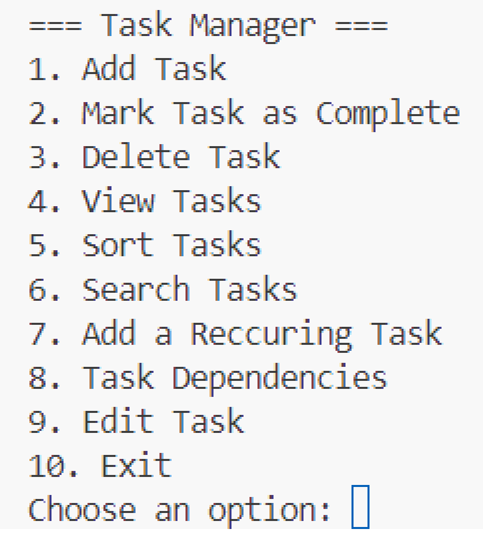
- Display the following data:
- Number of total tasks
- Number of tasks completed
- Number of tasks pending to be completed
- Completion rate
- Number of task in each task categories



6. Sample Input & Output

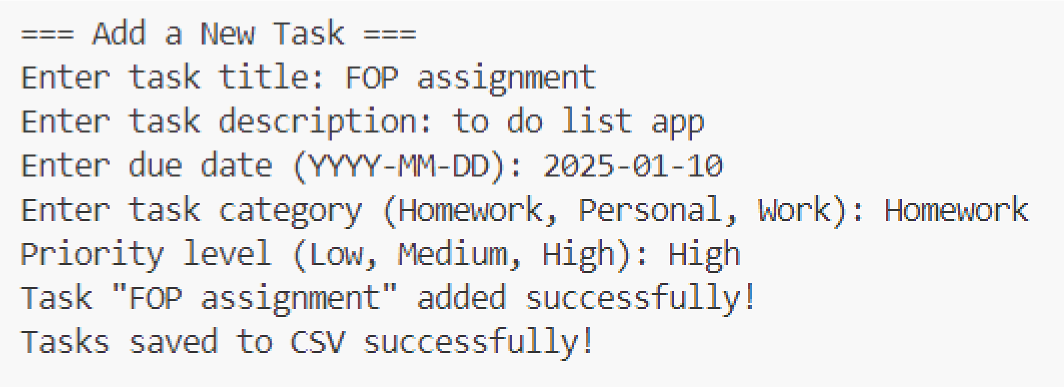
Basic Features

Terminal:



```
=== Task Manager ===
1. Add Task
2. Mark Task as Complete
3. Delete Task
4. View Tasks
5. Sort Tasks
6. Search Tasks
7. Add a Reccuring Task
8. Task Dependencies
9. Edit Task
10. Exit
Choose an option: 
```

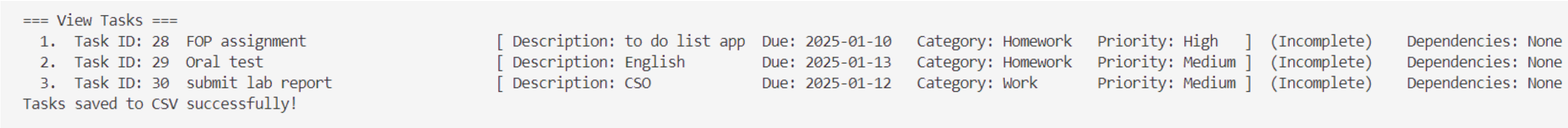
1. Add task



```
=== Add a New Task ===
Enter task title: FOP assignment
Enter task description: to do list app
Enter due date (YYYY-MM-DD): 2025-01-10
Enter task category (Homework, Personal, Work): Homework
Priority level (Low, Medium, High): High
Task "FOP assignment" added successfully!
Tasks saved to CSV successfully!
```

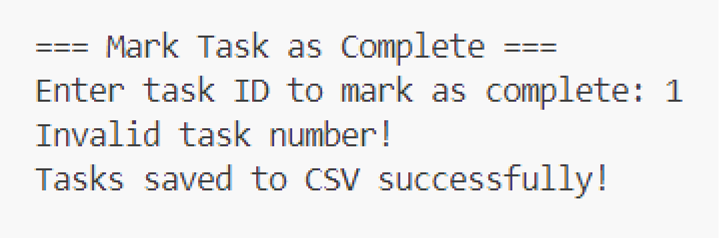
2. Task management

View tasks



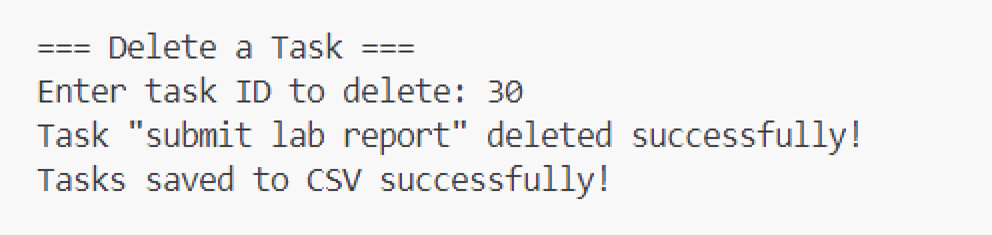
```
=== View Tasks ===
  1.  Task ID: 28  FOP assignment                      [ Description: to do list app  Due: 2025-01-10   Category: Homework   Priority: High   ] (Incomplete)    Dependencies: None
  2.  Task ID: 29  Oral test                           [ Description: English         Due: 2025-01-13   Category: Homework   Priority: Medium ] (Incomplete)    Dependencies: None
  3.  Task ID: 30  submit lab report                   [ Description: CSO             Due: 2025-01-12   Category: Work       Priority: Medium ] (Incomplete)    Dependencies: None
Tasks saved to CSV successfully!
```

Mark tasks as completed



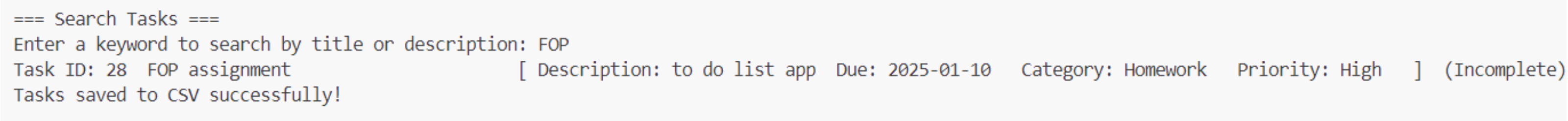
```
=== Mark Task as Complete ===
Enter task ID to mark as complete: 1
Invalid task number!
Tasks saved to CSV successfully!
```

3. Delete task



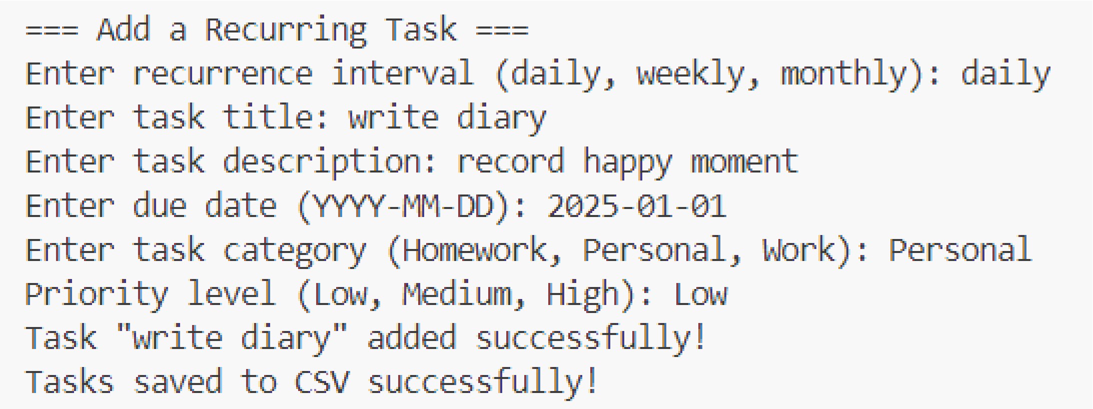
```
=== Delete a Task ===
Enter task ID to delete: 30
Task "submit lab report" deleted successfully!
Tasks saved to CSV successfully!
```

4. Search task



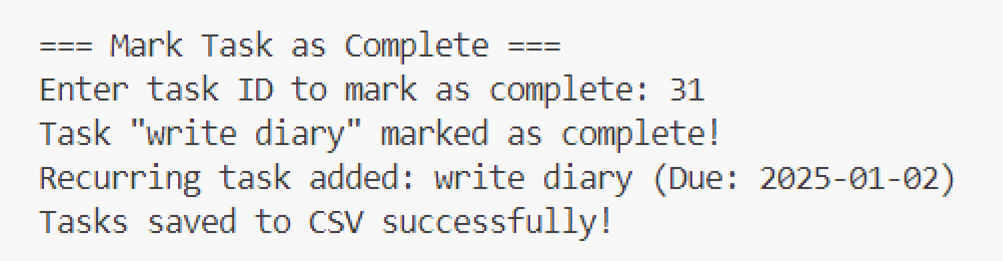
```
=== Search Tasks ===
Enter a keyword to search by title or description: FOP
Task ID: 28  FOP assignment                        [ Description: to do list app  Due: 2025-01-10   Category: Homework   Priority: High    ]  (Incomplete)
Tasks saved to CSV successfully!
```

5. Add recurring task

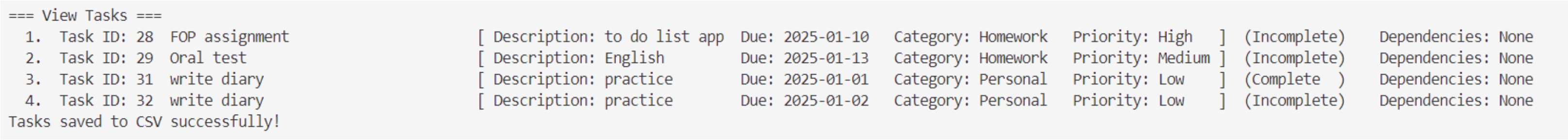


```
=== Add a Recurring Task ===
Enter recurrence interval (daily, weekly, monthly): daily
Enter task title: write diary
Enter task description: record happy moment
Enter due date (YYYY-MM-DD): 2025-01-01
Enter task category (Homework, Personal, Work): Personal
Priority level (Low, Medium, High): Low
Task "write diary" added successfully!
Tasks saved to CSV successfully!
```

When a recurring task is mark as completed, a same task with different due date will be automatically generated according to the recurrence interval



```
=== Mark Task as Complete ===
Enter task ID to mark as complete: 31
Task "write diary" marked as complete!
Recurring task added: write diary (Due: 2025-01-02)
Tasks saved to CSV successfully!
```

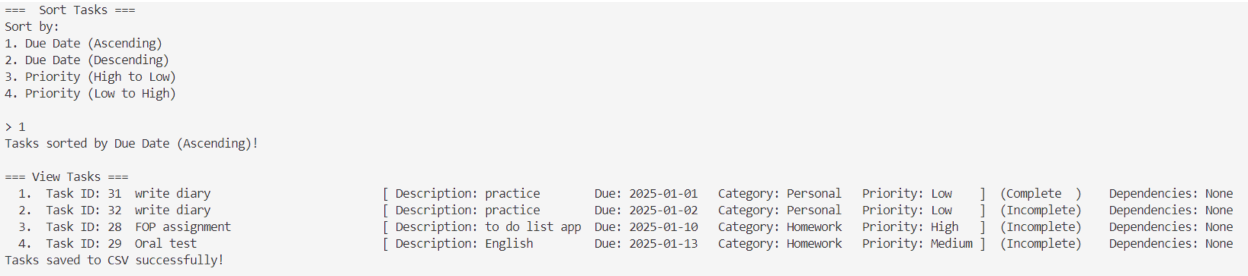


```
=== View Tasks ===
  1.  Task ID: 28  FOP assignment                        [ Description: to do list app  Due: 2025-01-10   Category: Homework   Priority: High   ]  (Incomplete)    Dependencies: None
  2.  Task ID: 29  Oral test                             [ Description: English         Due: 2025-01-13   Category: Homework   Priority: Medium ]  (Incomplete)    Dependencies: None
  3.  Task ID: 31  write diary                           [ Description: practice        Due: 2025-01-01   Category: Personal   Priority: Low    ]  (Complete  )    Dependencies: None
  4.  Task ID: 32  write diary                           [ Description: practice        Due: 2025-01-02   Category: Personal   Priority: Low    ]  (Incomplete)    Dependencies: None
Tasks saved to CSV successfully!
```

6. Sort task

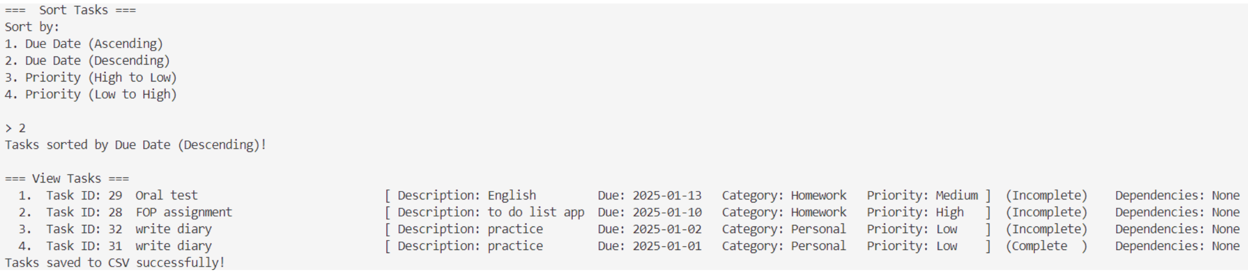
Sort task according to:

Ascending due date



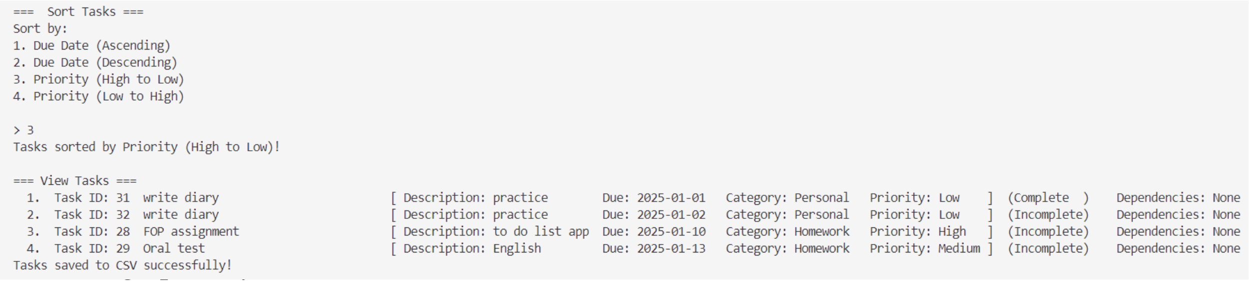
```
===  Sort Tasks ===
Sort by:
1. Due Date (Ascending)
2. Due Date (Descending)
3. Priority (High to Low)
4. Priority (Low to High)

> 1
Tasks sorted by Due Date (Ascending)!

=== View Tasks ===
  1.  Task ID: 31  write diary                          [ Description: practice        Due: 2025-01-01   Category: Personal   Priority: Low    ]  (Complete  )    Dependencies: None
  2.  Task ID: 32  write diary                          [ Description: practice        Due: 2025-01-02   Category: Personal   Priority: Low    ]  (Incomplete)    Dependencies: None
  3.  Task ID: 28  FOP assignment                       [ Description: to do list app  Due: 2025-01-10   Category: Homework   Priority: High   ]  (Incomplete)    Dependencies: None
  4.  Task ID: 29  Oral test                            [ Description: English         Due: 2025-01-13   Category: Homework   Priority: Medium ]  (Incomplete)    Dependencies: None
Tasks saved to CSV successfully!
```

Descending due date

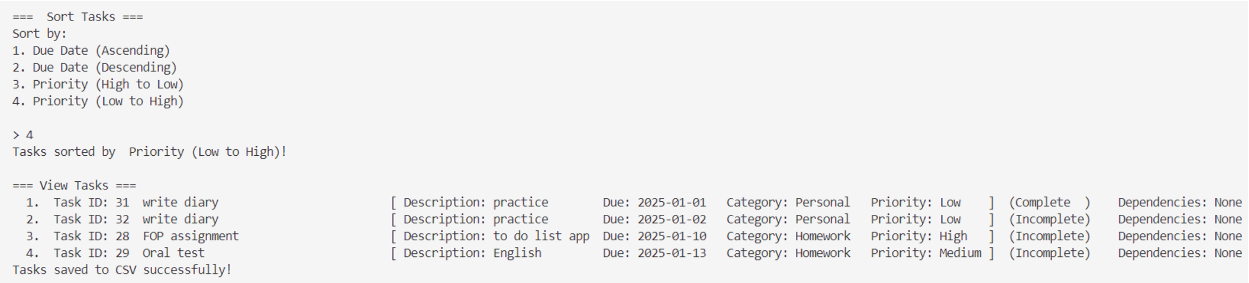


```
===  Sort Tasks ===
Sort by:
1. Due Date (Ascending)
2. Due Date (Descending)
3. Priority (High to Low)
4. Priority (Low to High)

> 2
Tasks sorted by Due Date (Descending)!

=== View Tasks ===
  1.  Task ID: 29  Oral test                            [ Description: English         Due: 2025-01-13   Category: Homework   Priority: Medium ]  (Incomplete)    Dependencies: None
  2.  Task ID: 28  FOP assignment                       [ Description: to do list app  Due: 2025-01-10   Category: Homework   Priority: High   ]  (Incomplete)    Dependencies: None
  3.  Task ID: 32  write diary                          [ Description: practice        Due: 2025-01-02   Category: Personal   Priority: Low    ]  (Incomplete)    Dependencies: None
  4.  Task ID: 31  write diary                          [ Description: practice        Due: 2025-01-01   Category: Personal   Priority: Low    ]  (Complete  )    Dependencies: None
Tasks saved to CSV successfully!
```

High to low priority

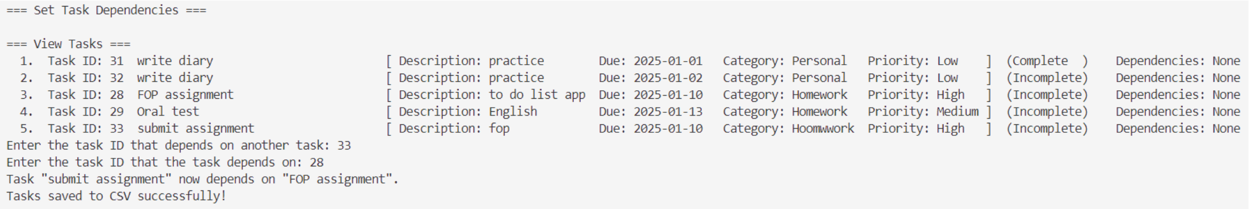


```
===  Sort Tasks ===
Sort by:
1. Due Date (Ascending)
2. Due Date (Descending)
3. Priority (High to Low)
4. Priority (Low to High)

> 3
Tasks sorted by Priority (High to Low)!

=== View Tasks ===
  1.  Task ID: 31  write diary                     [ Description: practice       Due: 2025-01-01   Category: Personal   Priority: Low    ]  (Complete  )    Dependencies: None
  2.  Task ID: 32  write diary                     [ Description: practice       Due: 2025-01-02   Category: Personal   Priority: Low    ]  (Incomplete)    Dependencies: None
  3.  Task ID: 28  FOP assignment                  [ Description: to do list app Due: 2025-01-10   Category: Homework   Priority: High   ]  (Incomplete)    Dependencies: None
  4.  Task ID: 29  Oral test                       [ Description: English        Due: 2025-01-13   Category: Homework   Priority: Medium ]  (Incomplete)    Dependencies: None
Tasks saved to CSV successfully!
```



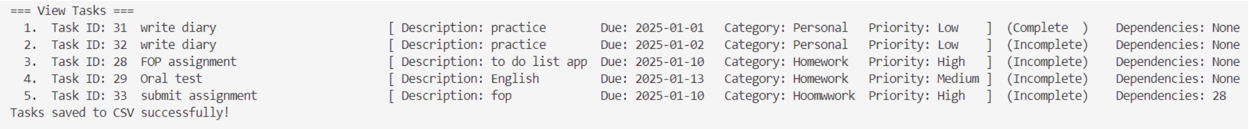
```
===  Sort Tasks ===
Sort by:
1. Due Date (Ascending)
2. Due Date (Descending)
3. Priority (High to Low)
4. Priority (Low to High)

> 4
Tasks sorted by  Priority (Low to High)!

=== View Tasks ===
  1.  Task ID: 31  write diary                     [ Description: practice       Due: 2025-01-01   Category: Personal   Priority: Low    ]  (Complete  )    Dependencies: None
  2.  Task ID: 32  write diary                     [ Description: practice       Due: 2025-01-02   Category: Personal   Priority: Low    ]  (Incomplete)    Dependencies: None
  3.  Task ID: 28  FOP assignment                  [ Description: to do list app Due: 2025-01-10   Category: Homework   Priority: High   ]  (Incomplete)    Dependencies: None
  4.  Task ID: 29  Oral test                       [ Description: English        Due: 2025-01-13   Category: Homework   Priority: Medium ]  (Incomplete)    Dependencies: None
Tasks saved to CSV successfully!
```

7. Add task dependencies



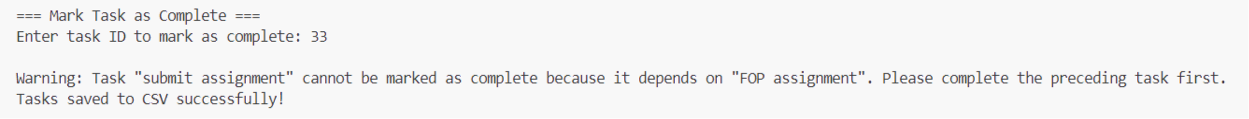
```
=== Set Task Dependencies ===

=== View Tasks ===
  1.  Task ID: 31  write diary                     [ Description: practice       Due: 2025-01-01   Category: Personal   Priority: Low    ]  (Complete  )    Dependencies: None
  2.  Task ID: 32  write diary                     [ Description: practice       Due: 2025-01-02   Category: Personal   Priority: Low    ]  (Incomplete)    Dependencies: None
  3.  Task ID: 28  FOP assignment                  [ Description: to do list app Due: 2025-01-10   Category: Homework   Priority: High   ]  (Incomplete)    Dependencies: None
  4.  Task ID: 29  Oral test                       [ Description: English        Due: 2025-01-13   Category: Homework   Priority: Medium ]  (Incomplete)    Dependencies: None
  5.  Task ID: 33  submit assignment               [ Description: fop            Due: 2025-01-10   Category: Hoomwork   Priority: High   ]  (Incomplete)    Dependencies: None
Enter the task ID that depends on another task: 33
Enter the task ID that the task depends on: 28
Task "submit assignment" now depends on "FOP assignment".
Tasks saved to CSV successfully!
```

The task ID that the task depends on will be shown in view task



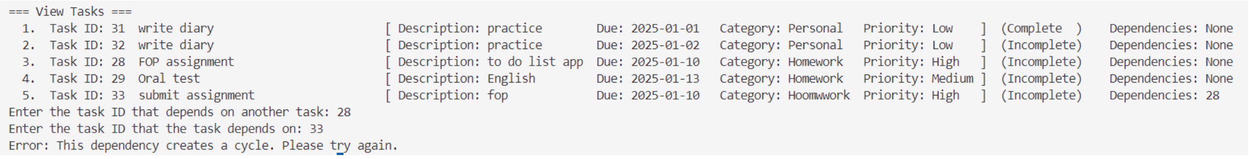
```
=== View Tasks ===
  1.  Task ID: 31  write diary                     [ Description: practice       Due: 2025-01-01   Category: Personal   Priority: Low    ]  (Complete  )    Dependencies: None
  2.  Task ID: 32  write diary                     [ Description: practice       Due: 2025-01-02   Category: Personal   Priority: Low    ]  (Incomplete)    Dependencies: None
  3.  Task ID: 28  FOP assignment                  [ Description: to do list app Due: 2025-01-10   Category: Homework   Priority: High   ]  (Incomplete)    Dependencies: None
  4.  Task ID: 29  Oral test                       [ Description: English        Due: 2025-01-13   Category: Homework   Priority: Medium ]  (Incomplete)    Dependencies: None
  5.  Task ID: 33  submit assignment               [ Description: fop            Due: 2025-01-10   Category: Hoomwork   Priority: High   ]  (Incomplete)    Dependencies: 28
Tasks saved to CSV successfully!
```

If the user attempts to mark a dependent task complete before its preceding task, a warning message is shown.



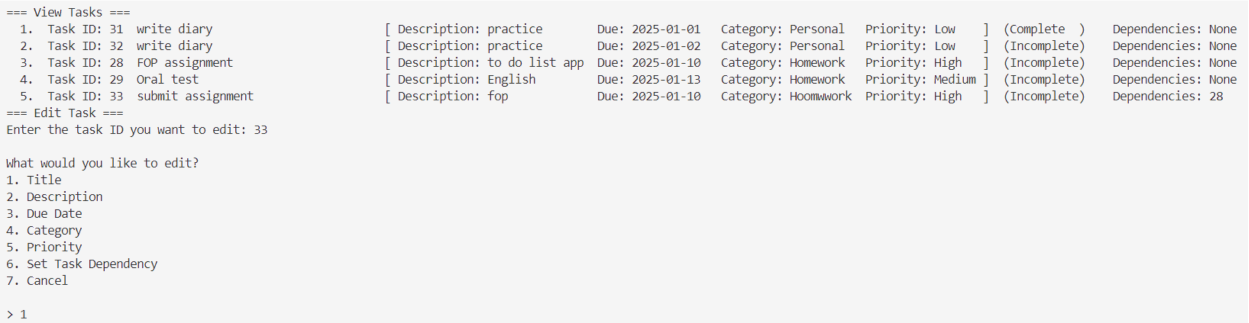
```
=== Mark Task as Complete ===
Enter task ID to mark as complete: 33

Warning: Task "submit assignment" cannot be marked as complete because it depends on "FOP assignment". Please complete the preceding task first.
Tasks saved to CSV successfully!
```

A cycle of dependency is not allowed. If users attempt to create a dependency cycle, a warning message is shown.

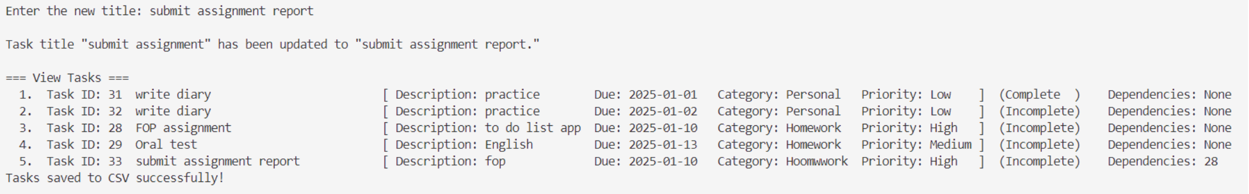


```
=== View Tasks ===
  1.  Task ID: 31  write diary                         [ Description: practice        Due: 2025-01-01   Category: Personal   Priority: Low    ]  (Complete  )    Dependencies: None
  2.  Task ID: 32  write diary                         [ Description: practice        Due: 2025-01-02   Category: Personal   Priority: Low    ]  (Incomplete)    Dependencies: None
  3.  Task ID: 28  FOP assignment                      [ Description: to do list app  Due: 2025-01-10   Category: Homework   Priority: High   ]  (Incomplete)    Dependencies: None
  4.  Task ID: 29  Oral test                           [ Description: English         Due: 2025-01-13   Category: Homework   Priority: Medium ]  (Incomplete)    Dependencies: None
  5.  Task ID: 33  submit assignment                   [ Description: fop             Due: 2025-01-10   Category: Hoomwwork  Priority: High   ]  (Incomplete)    Dependencies: 28
Enter the task ID that depends on another task: 28
Enter the task ID that the task depends on: 33
Error: This dependency creates a cycle. Please try again.
```

8. Edit task

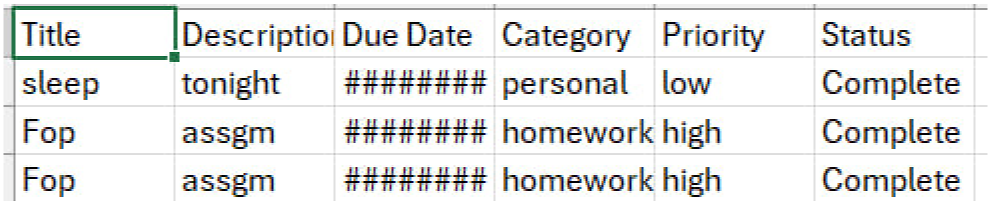


```
=== View Tasks ===
  1.  Task ID: 31  write diary                       [ Description: practice        Due: 2025-01-01   Category: Personal   Priority: Low    ]  (Complete  )    Dependencies: None
  2.  Task ID: 32  write diary                       [ Description: practice        Due: 2025-01-02   Category: Personal   Priority: Low    ]  (Incomplete)    Dependencies: None
  3.  Task ID: 28  FOP assignment                    [ Description: to do list app  Due: 2025-01-10   Category: Homework   Priority: High   ]  (Incomplete)    Dependencies: None
  4.  Task ID: 29  Oral test                         [ Description: English         Due: 2025-01-13   Category: Homework   Priority: Medium ]  (Incomplete)    Dependencies: None
  5.  Task ID: 33  submit assignment                 [ Description: fop             Due: 2025-01-10   Category: Hoomwwork  Priority: High   ]  (Incomplete)    Dependencies: 28
=== Edit Task ===
Enter the task ID you want to edit: 33

What would you like to edit?
1. Title
2. Description
3. Due Date
4. Category
5. Priority
6. Set Task Dependency
7. Cancel

> 1
```



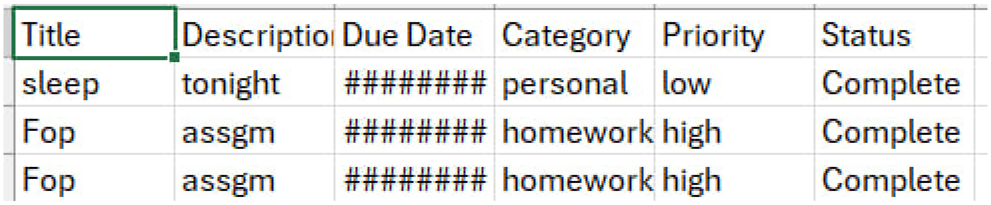
```
Enter the new title: submit assignment report

Task title "submit assignment" has been updated to "submit assignment report."

=== View Tasks ===
  1.  Task ID: 31  write diary                       [ Description: practice        Due: 2025-01-01   Category: Personal   Priority: Low    ]  (Complete  )    Dependencies: None
  2.  Task ID: 32  write diary                       [ Description: practice        Due: 2025-01-02   Category: Personal   Priority: Low    ]  (Incomplete)    Dependencies: None
  3.  Task ID: 28  FOP assignment                    [ Description: to do list app  Due: 2025-01-10   Category: Homework   Priority: High   ]  (Incomplete)    Dependencies: None
  4.  Task ID: 29  Oral test                         [ Description: English         Due: 2025-01-13   Category: Homework   Priority: Medium ]  (Incomplete)    Dependencies: None
  5.  Task ID: 33  submit assignment report          [ Description: fop             Due: 2025-01-10   Category: Hoomwwork  Priority: High   ]  (Incomplete)    Dependencies: 28
Tasks saved to CSV successfully!
```

9. Load tasks from CSV file



Title	Descriptio	Due Date	Category	Priority	Status
sleep	tonight	########	personal	low	Complete
Fop	assgm	########	homework	high	Complete
Fop	assgm	########	homework	high	Complete

10. Save task to CSV file

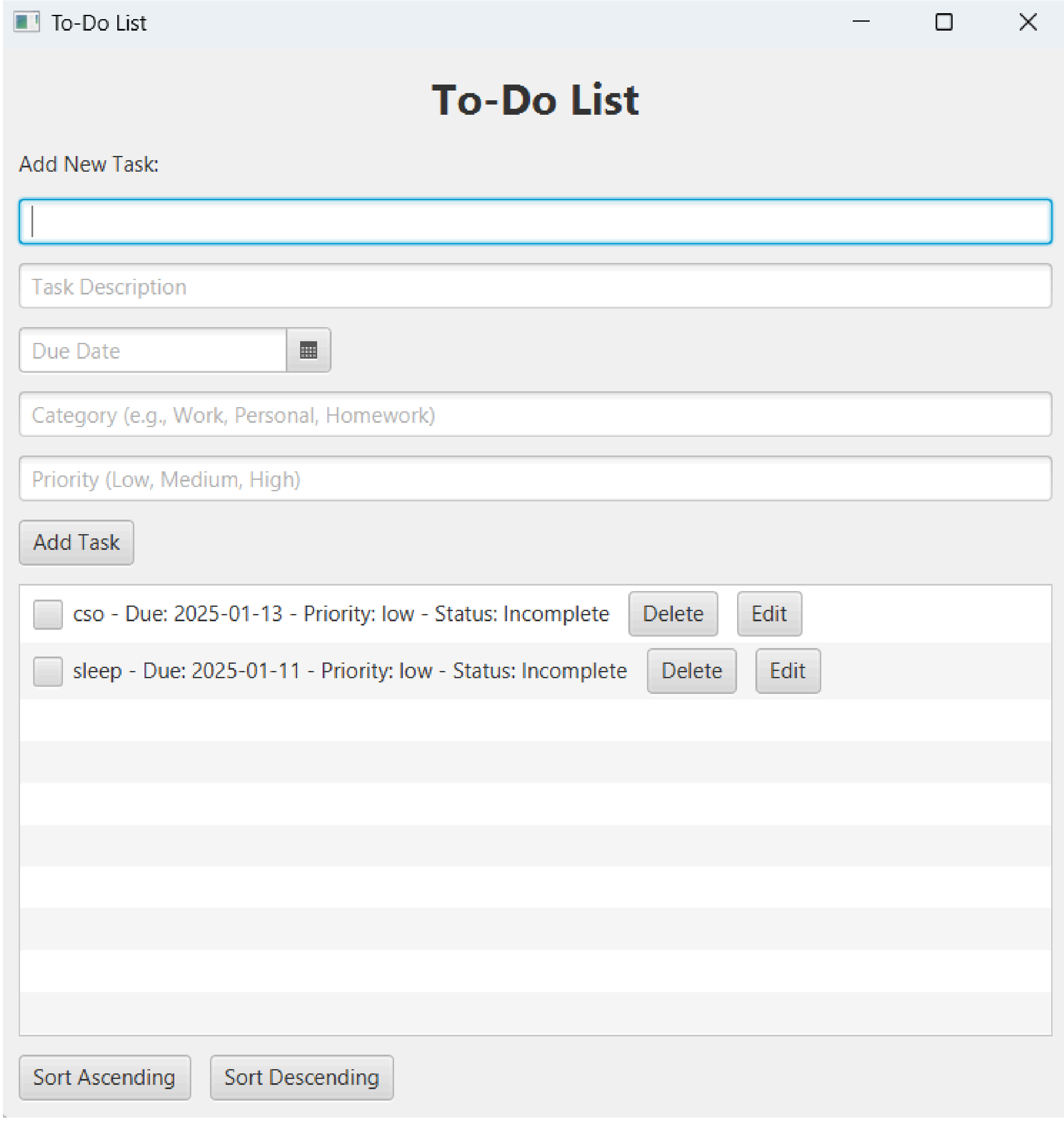


Title	Descriptio	Due Date	Category	Priority	Status
sleep	tonight	########	personal	low	Complete
Fop	assgm	########	homework	high	Complete
Fop	assgm	########	homework	high	Complete

Extra Features

1. Graphical User Interface

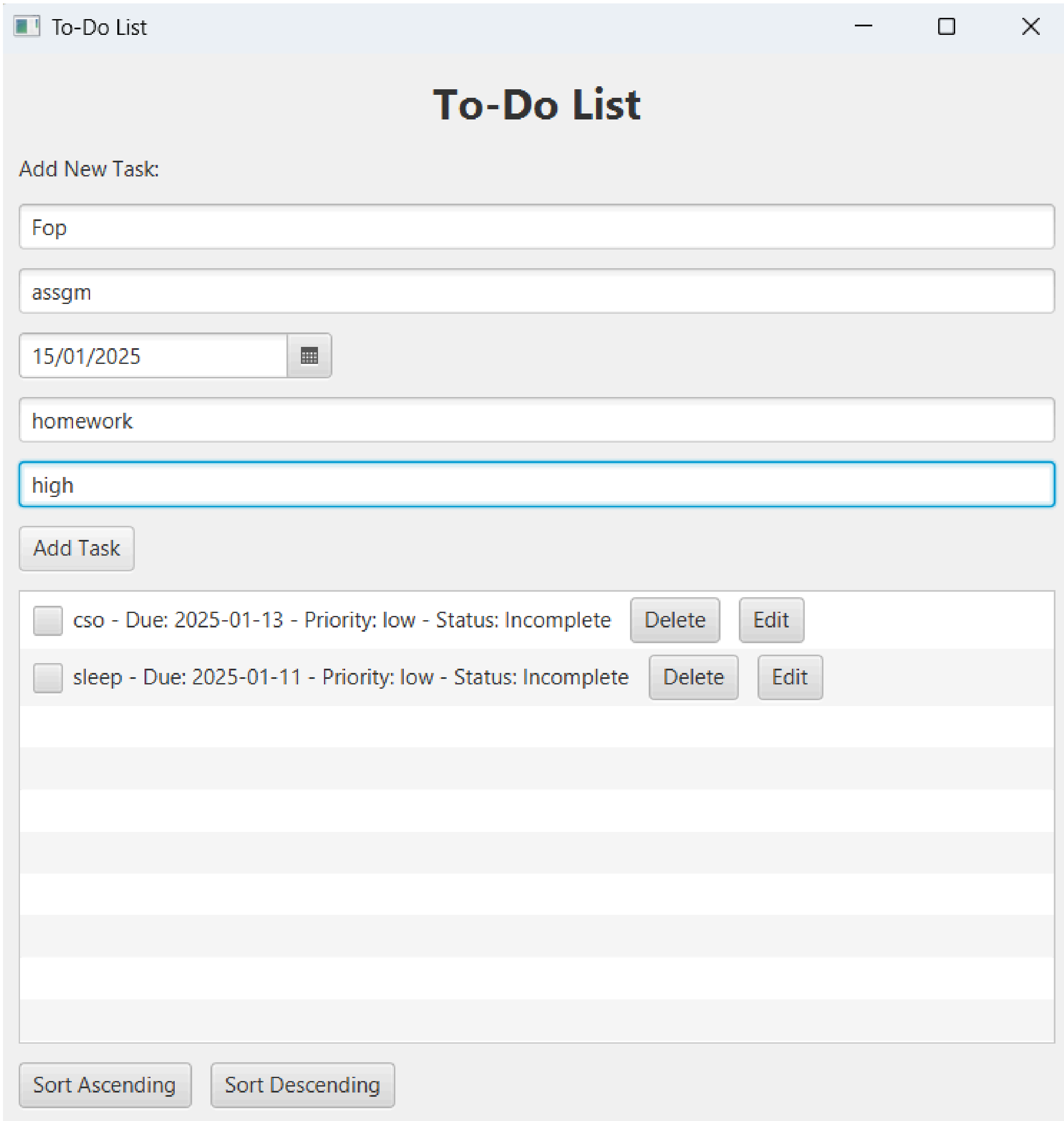
Main page:



□ cso - Due: 2025-01-13 - Priority: low - Status: Incomplete

□ sleep - Due: 2025-01-11 - Priority: low - Status: Incomplete

Add Task by filling in text field and then press “Add Task” button**:**



To-Do List

Add New Task:

Fop

assgm

15/01/2025

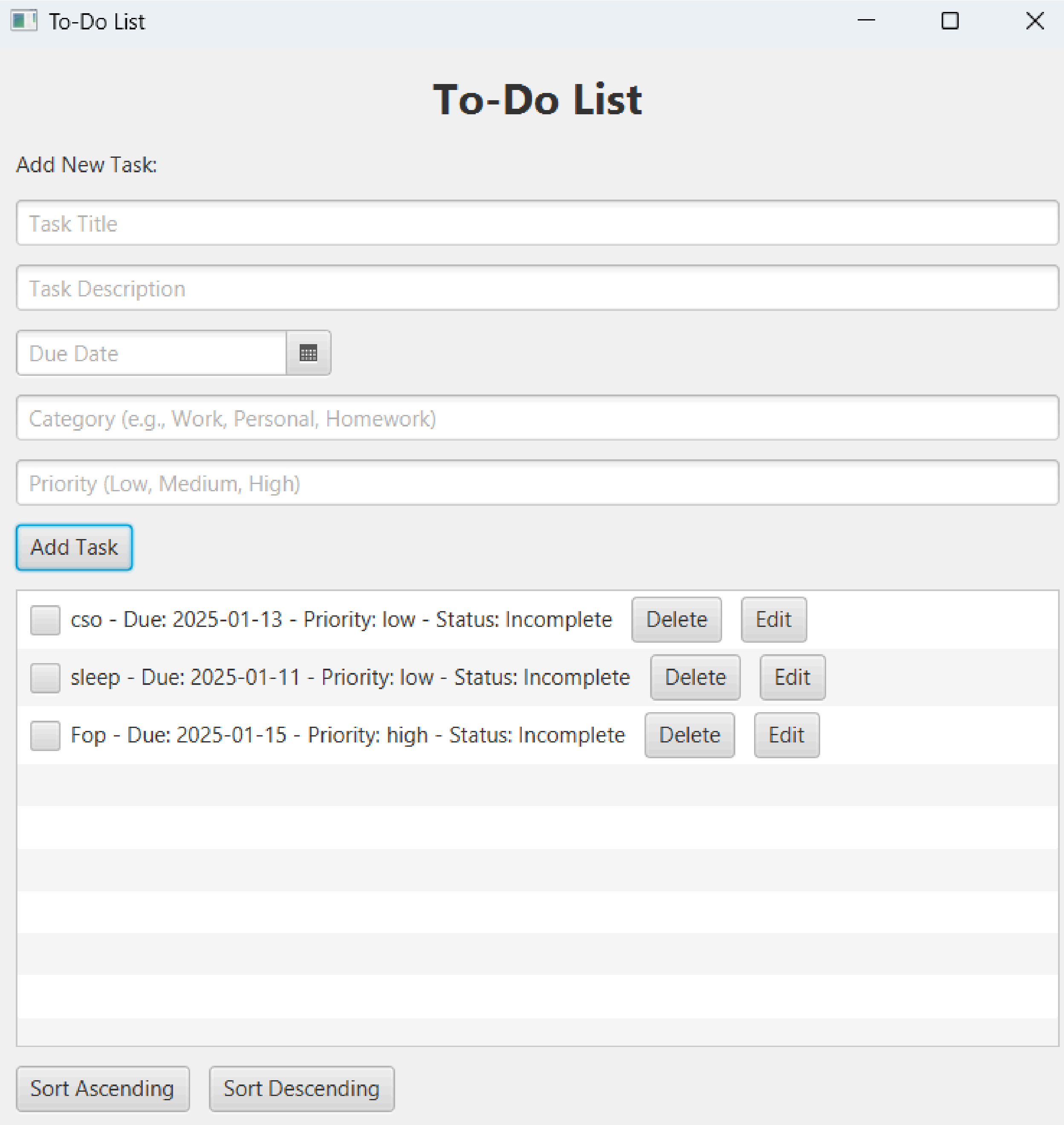
homework

high

□ cso - Due: 2025-01-13 - Priority: low - Status: Incomplete

□ sleep - Due: 2025-01-11 - Priority: low - Status: Incomplete

After pressing the “**Add Task**” button, the task will display on the **list view**:



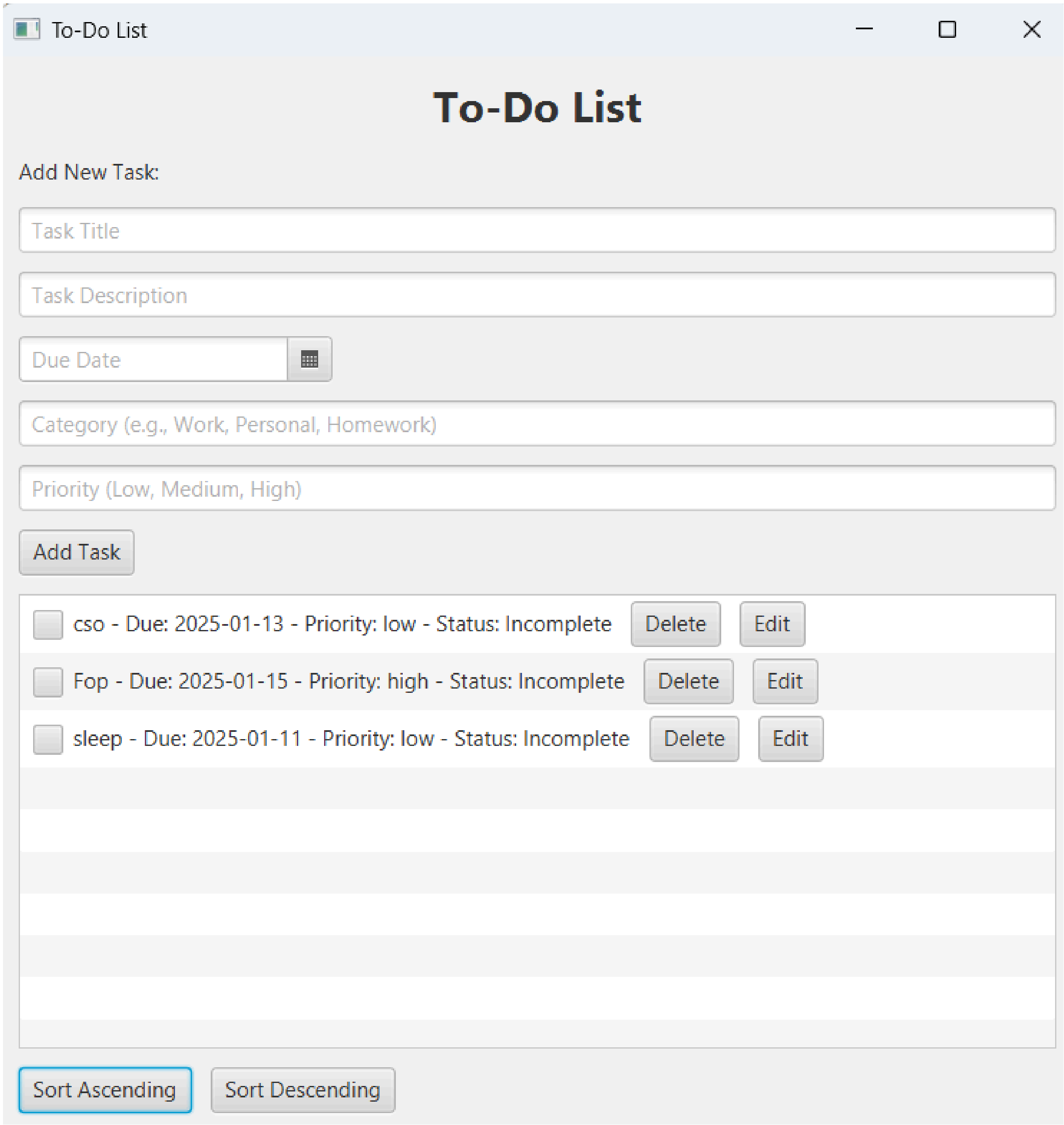
□ cso - Due: 2025-01-13 - Priority: low - Status: Incomplete

□ sleep - Due: 2025-01-11 - Priority: low - Status: Incomplete

□ Fop - Due: 2025-01-15 - Priority: high - Status: Incomplete

Sort Task:

Ascending:

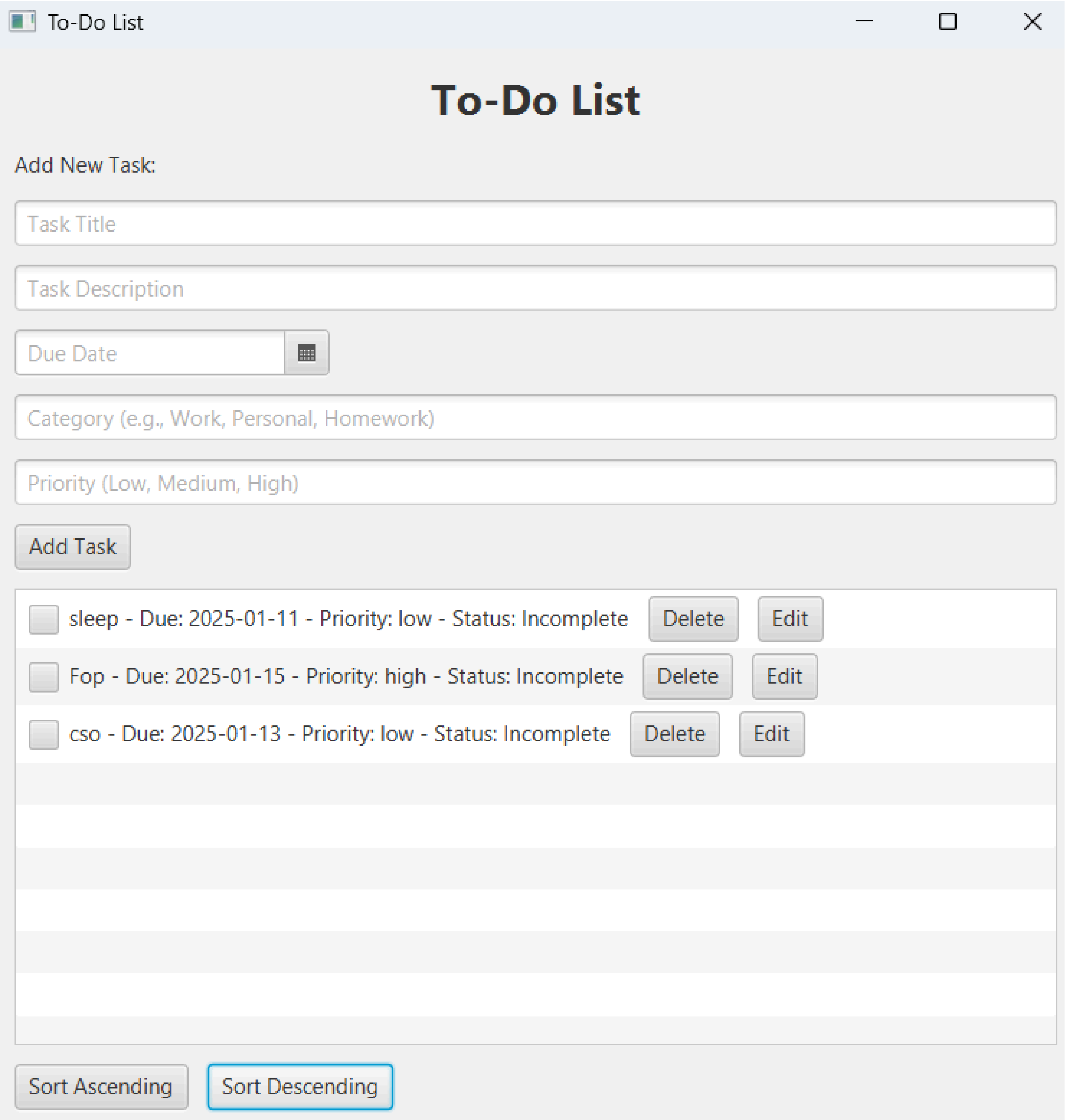


□ cso - Due: 2025-01-13 - Priority: low - Status: Incomplete

□ Fop - Due: 2025-01-15 - Priority: high - Status: Incomplete

□ sleep - Due: 2025-01-11 - Priority: low - Status: Incomplete

Descending:

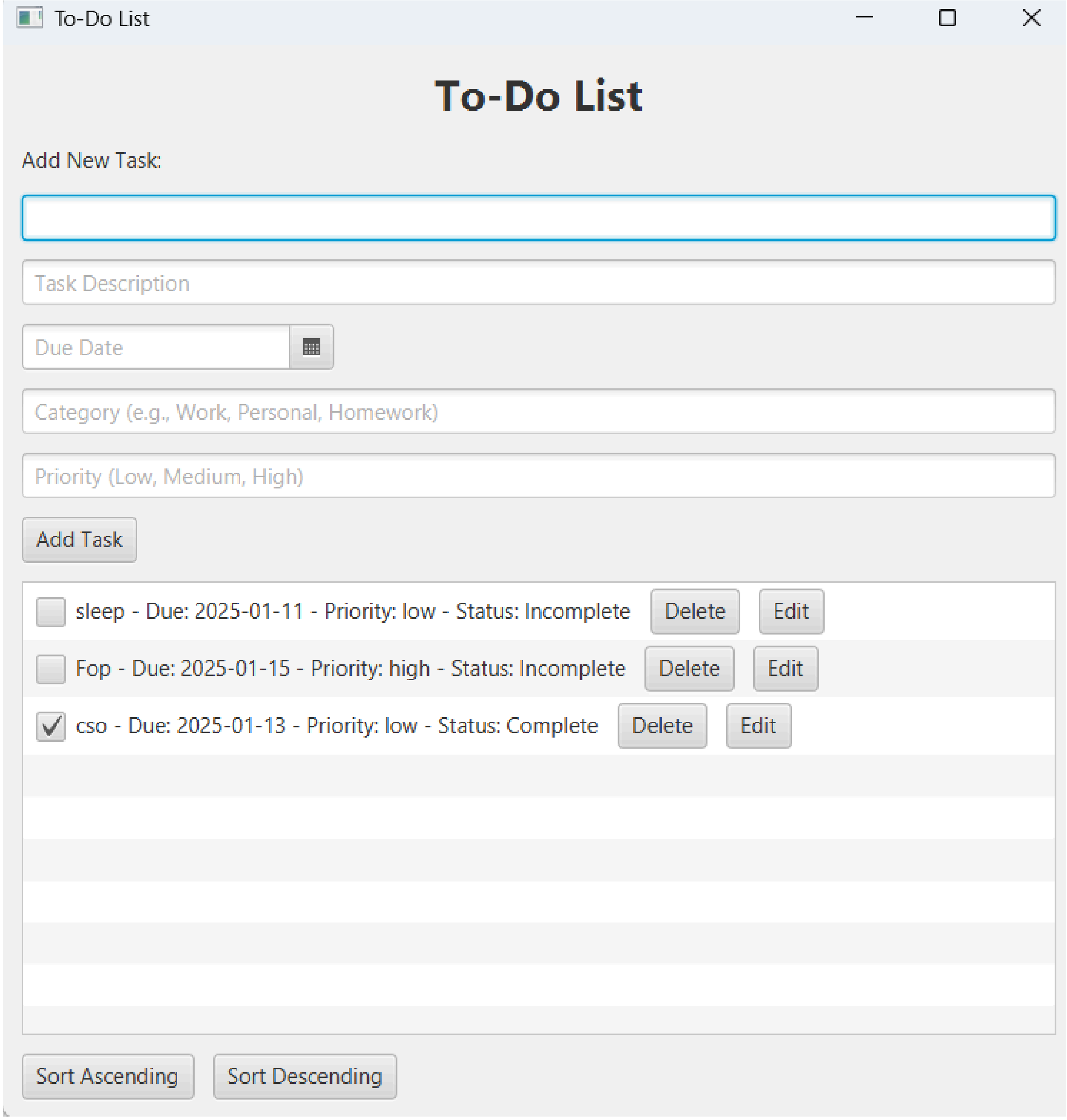


□ sleep - Due: 2025-01-11 - Priority: low - Status: Incomplete

□ Fop - Due: 2025-01-15 - Priority: high - Status: Incomplete

□ cso - Due: 2025-01-13 - Priority: low - Status: Incomplete

Mark as Complete by put a tick at the checkbox (**Status: Complete):**



To-Do List

Add New Task:

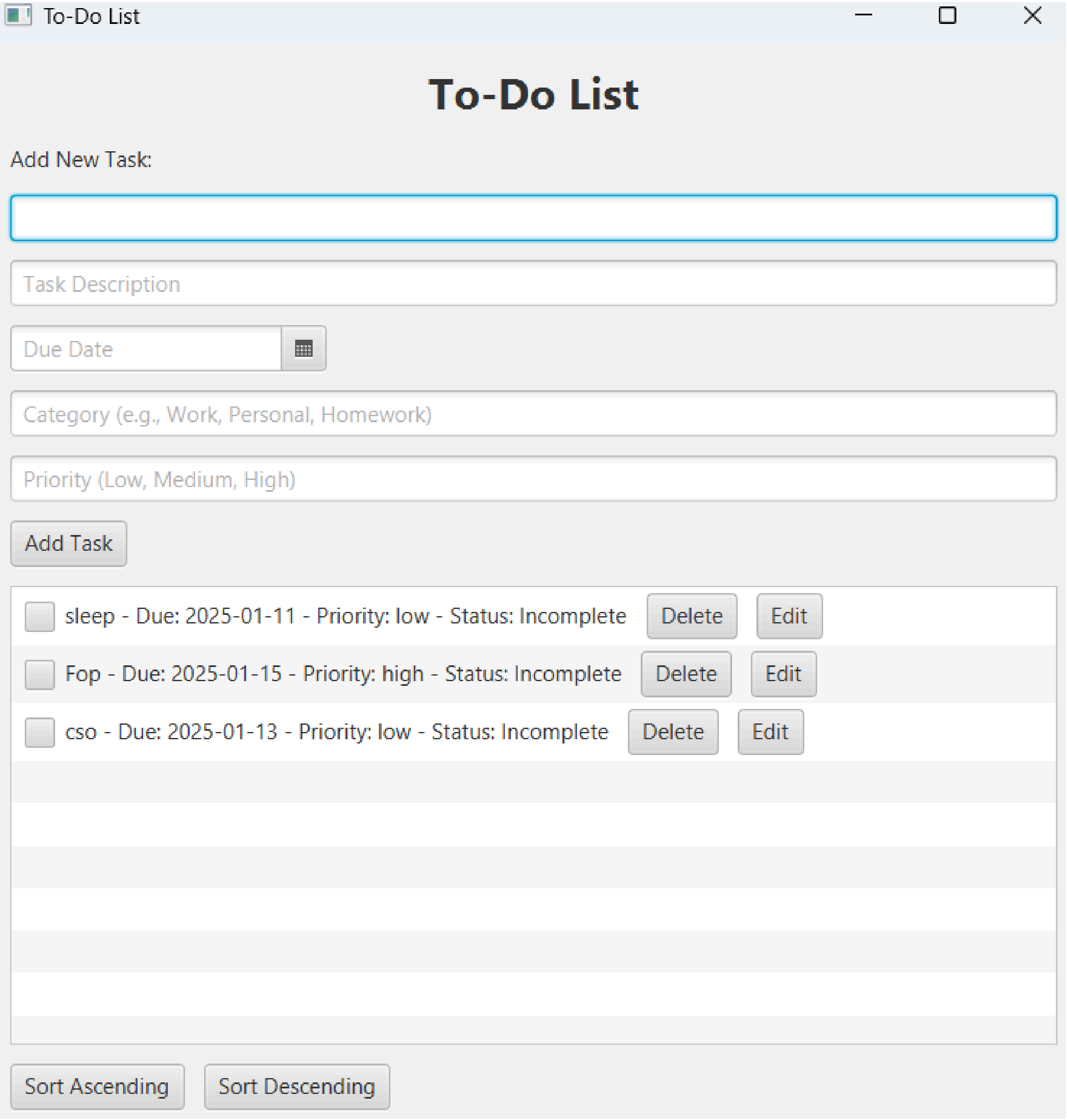
Task Description
Due Date
Category (e.g., Work, Personal, Homework)
Priority (Low, Medium, High)
Add Task

□ sleep - Due: 2025-01-11 - Priority: low - Status: Incomplete

□ Fop - Due: 2025-01-15 - Priority: high - Status: Incomplete

☑ cso - Due: 2025-01-13 - Priority: low - Status: Complete

If the user remove the tick, the status become **incomplete**:



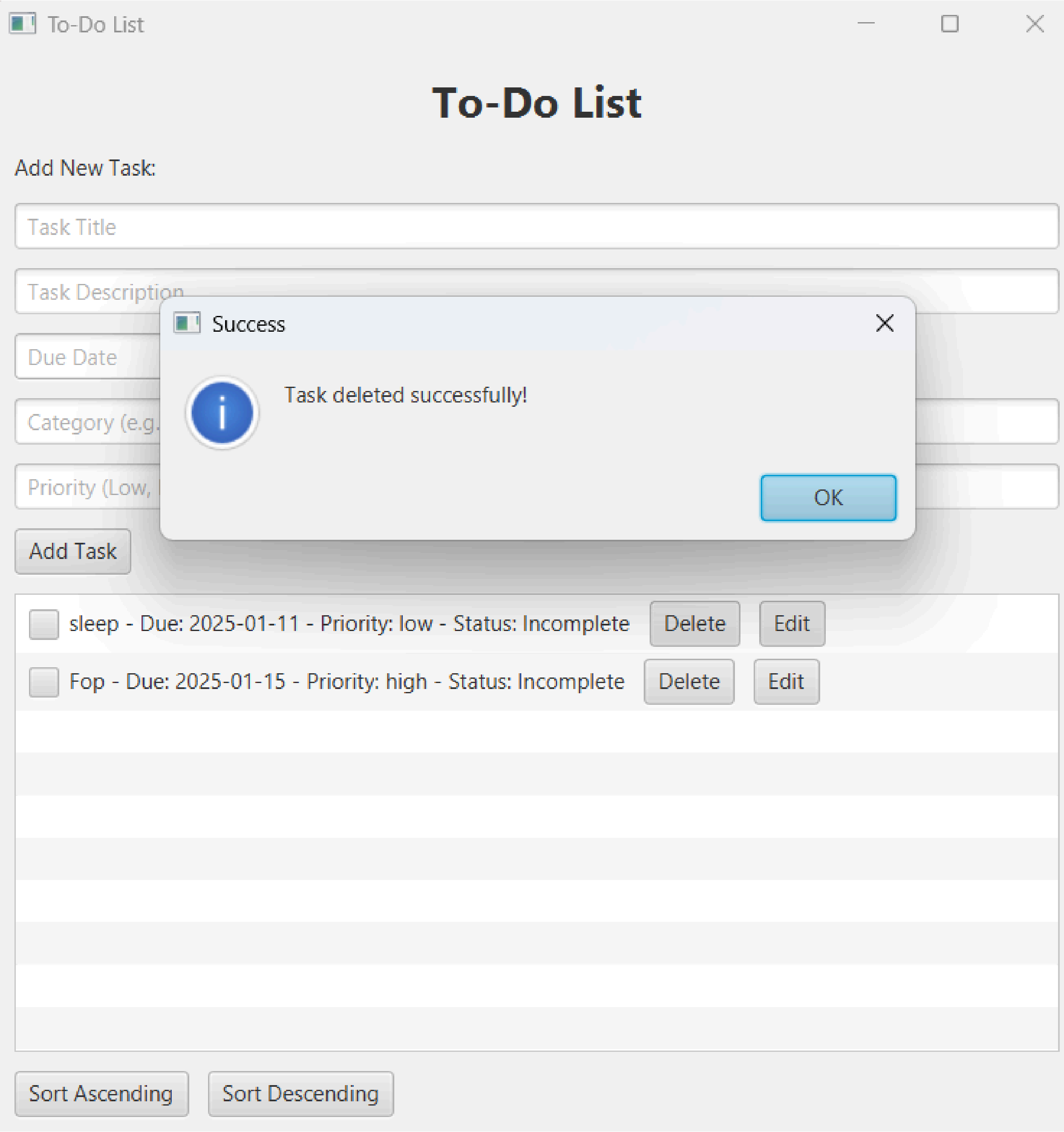
To-Do List
To-Do List
Add New Task:
Task Description
Due Date
Category (e.g., Work, Personal, Homework)
Priority (Low, Medium, High)
Add Task

□ sleep - Due: 2025-01-11 - Priority: low - Status: Incomplete

□ Fop - Due: 2025-01-15 - Priority: high - Status: Incomplete

□ cso - Due: 2025-01-13 - Priority: low - Status: Incomplete

Delete Task by clicking the “Delete” button:

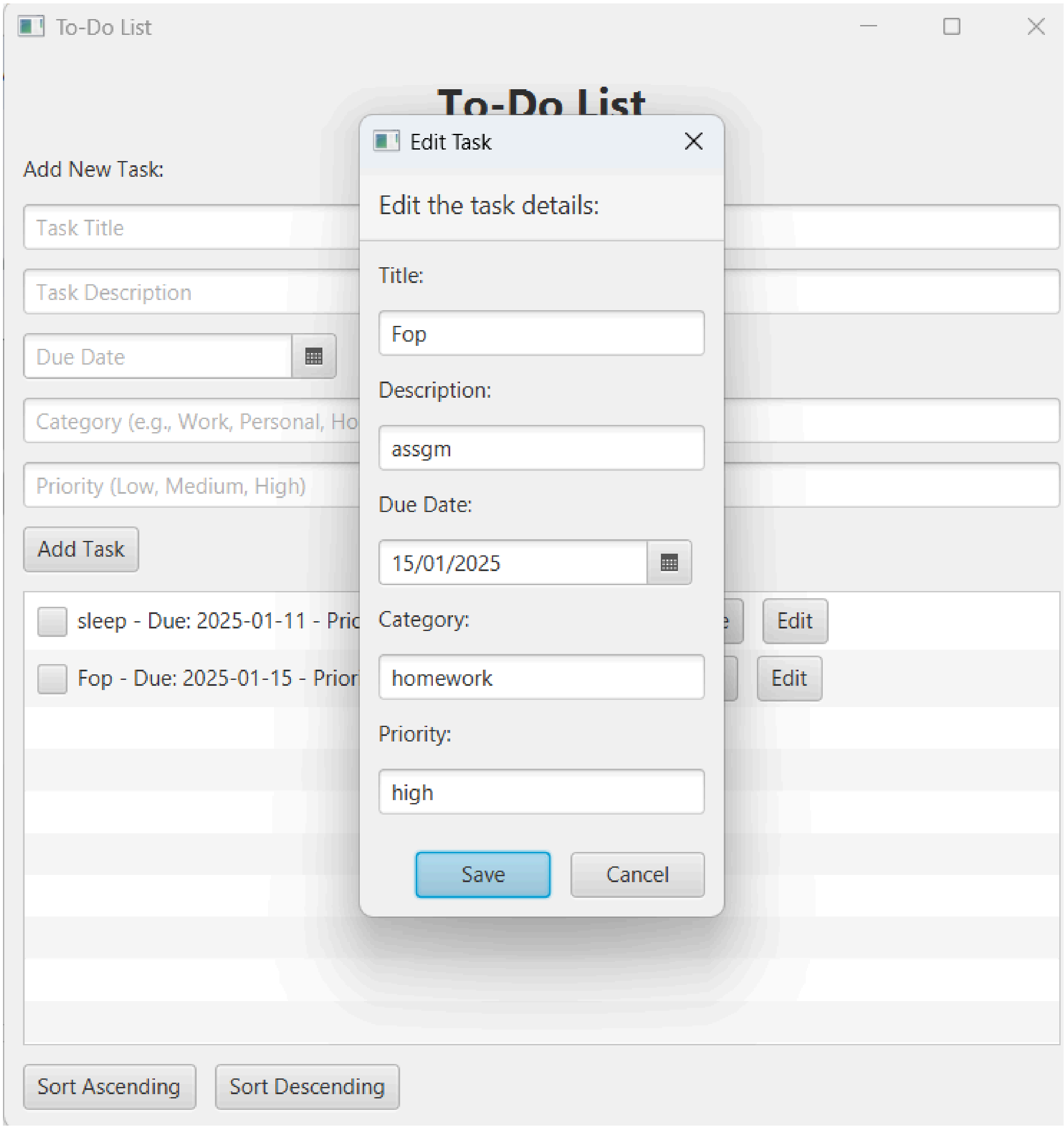


Task deleted successfully!

□ sleep - Due: 2025-01-11 - Priority: low - Status: Incomplete

□ Fop - Due: 2025-01-15 - Priority: high - Status: Incomplete

Edit task by clicking the “Edit” button and press “**Save**” button after editing the task:



sleep - Due: 2025-01-11 - Pric

Fop - Due: 2025-01-15 - Prior

Edit the task details:

Title: Fop

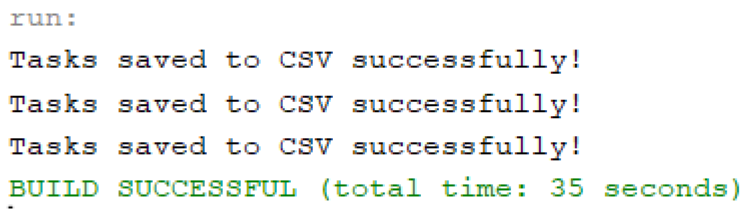
Description: assgm

Due Date: 15/01/2025

Category: homework

Priority: high

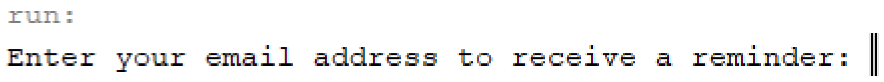
All modification in the GUI will be saved to CSV file



```
run:
Tasks saved to CSV successfully!
Tasks saved to CSV successfully!
Tasks saved to CSV successfully!
BUILD SUCCESSFUL (total time: 35 seconds)
```

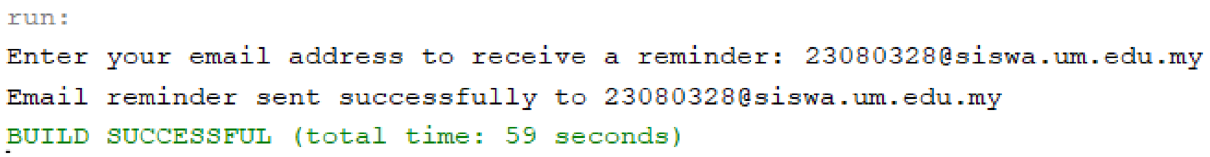
2. Email Notification System

Prompt user's email:

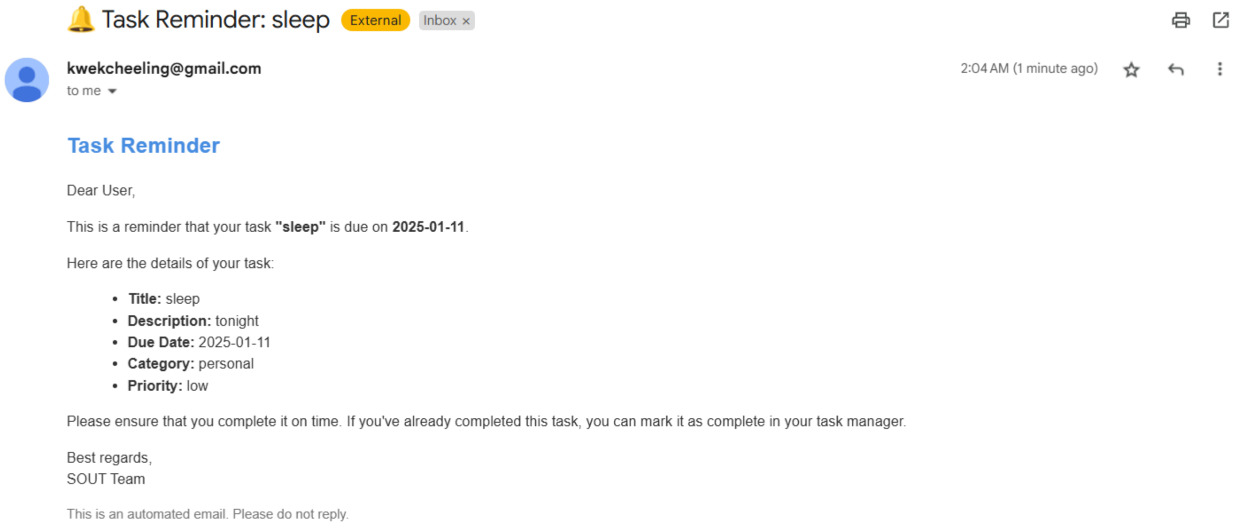


```
run:
Enter your email address to receive a reminder: 
```

If the email is sent successfully:



```
run:
Enter your email address to receive a reminder: 23080328@siswa.um.edu.my
Email reminder sent successfully to 23080328@siswa.um.edu.my
BUILD SUCCESSFUL (total time: 59 seconds)
```



🔔 Task Reminder: sleep

kwekcheeling@gmail.com
to me

2:04 AM (1 minute ago)

Task Reminder

Dear User,

This is a reminder that your task **"sleep"** is due on **2025-01-11**.

Here are the details of your task:

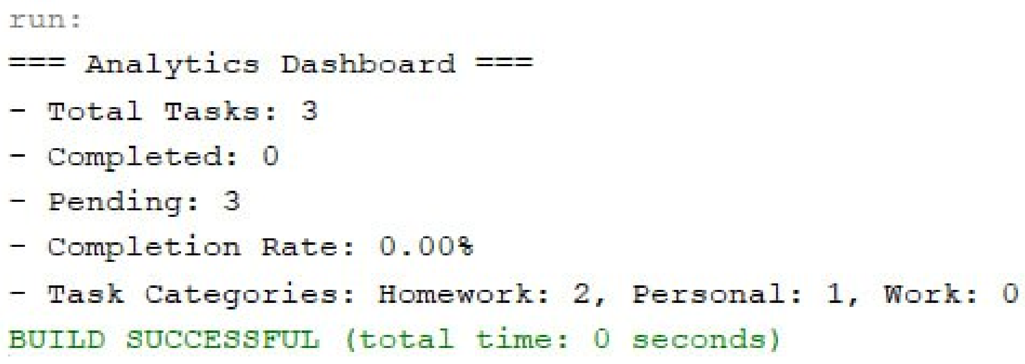
- **Title:** sleep
- **Description:** tonight
- **Due Date:** 2025-01-11
- **Category:** personal
- **Priority:** low

Please ensure that you complete it on time. If you've already completed this task, you can mark it as complete in your task manager.

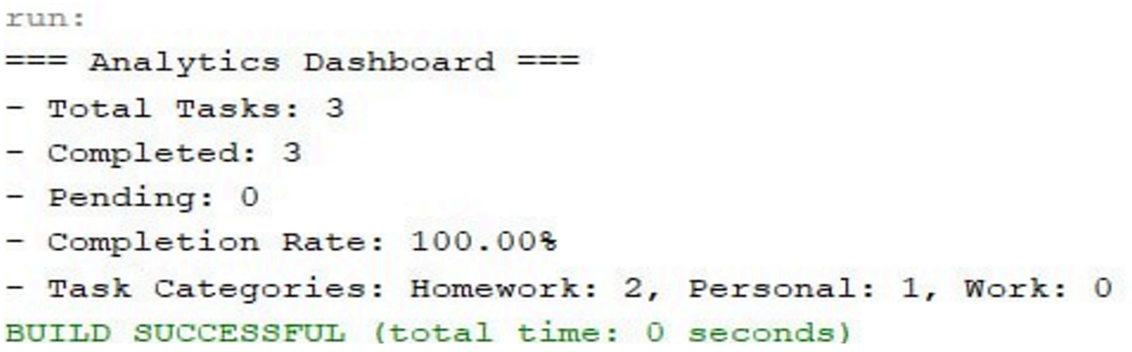
Best regards,
SOUT Team

This is an automated email. Please do not reply.

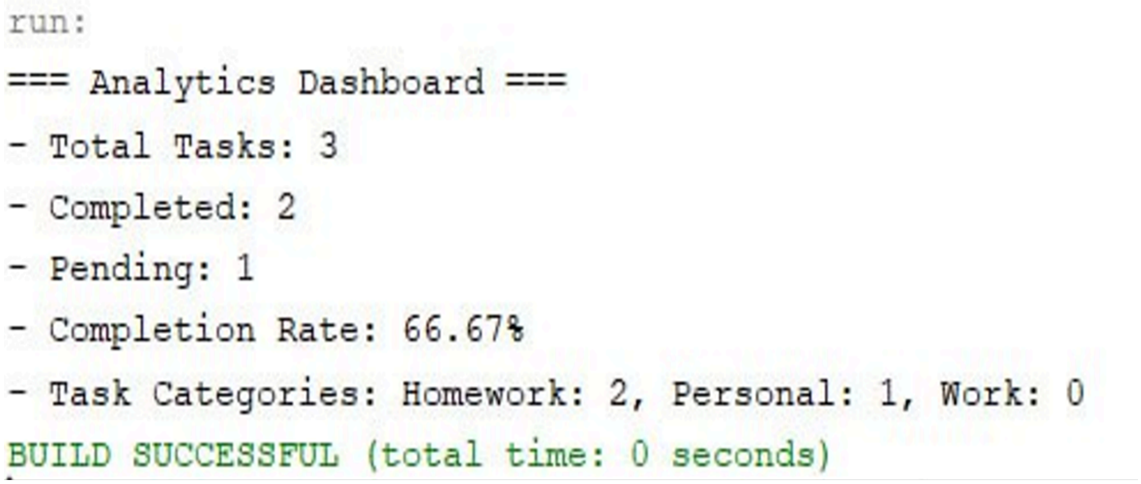
3. Data Analytics



```
run:
=== Analytics Dashboard ===
- Total Tasks: 3
- Completed: 0
- Pending: 3
- Completion Rate: 0.00%
- Task Categories: Homework: 2, Personal: 1, Work: 0
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
run:
=== Analytics Dashboard ===
- Total Tasks: 3
- Completed: 3
- Pending: 0
- Completion Rate: 100.00%
- Task Categories: Homework: 2, Personal: 1, Work: 0
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
run:
=== Analytics Dashboard ===
- Total Tasks: 3
- Completed: 2
- Pending: 1
- Completion Rate: 66.67%
- Task Categories: Homework: 2, Personal: 1, Work: 0
BUILD SUCCESSFUL (total time: 0 seconds)
```