

**Open-Ended Lab Report**

**Course Code:** CSE 2104

**Course Title:** Object Oriented Programming (CSE2104)

**Submitted By:** Md. Tazminur Rahman Tanim

**ID:** 242014124

**Section:** 4L

**Submitted By:** Task Management System (GUI Based)

**Submitted To:** Towsif Zahin Khan

**Lecturer, Department Of Computer Science And Engineering**

**Date of Submission: Aug 24, 2025**

**Objectives of the Software :**

The primary objectives of this Task Management System are:

* To design and implement a **GUI-based task management application** in Java.
* To demonstrate the **core principles of Object-Oriented Programming** (Encapsulation, Inheritance, Polymorphism, Abstraction).
* To integrate **file handling** for storing and retrieving task data persistently in tasks.csv.
* To allow users to perform **basic CRUD operations** (Add, Update, Delete, View Tasks).
* To provide a **user-friendly graphical interface** using Java Swing.
* To ensure the project aligns with the **open-ended lab requirements** of using modern tools, OOP, and GUI.

**System functionalities and features:**

The **Task Management System (GUI-based)** incorporates the following features:

* **Add Task**: Users can add tasks with title, description, due date, and priority.
* **Update Task**: Existing tasks can be selected from the table and updated.
* **Delete Task**: Users can delete any task after confirmation.
* **View Tasks**: A table displays all tasks in a structured format.
* **Persistence (File Handling)**: Tasks are stored in tasks.csv and automatically loaded at startup.
* **GUI (Java Swing)**: User-friendly interface with forms, buttons, and a table.
* **OOP Principles Implemented**:
  + Encapsulation (private fields with getters/setters in Task)
  + Inheritance (ImportantTask extends Task)
  + Polymorphism (overridden toString() method)
  + Abstraction (TaskOperations interface)

This ensures the software is both **functional** and **aligned with real-world task management needs**.

## **Design structured algorithms :**

### **Algorithm for Adding a Task**

1. Start
2. Accept input: Title, Description, Due Date, Priority
3. Create a new Task object with these details
4. Append the task into the ArrayList of tasks in TaskManager
5. Save tasks into tasks.csv using file handling
6. Refresh the GUI table to show the new task
7. End

### **Algorithm for Updating a Task**

1. Start
2. User selects a task from the GUI table
3. Accept updated values from input fields
4. Replace the old Task object with a new one at the selected index
5. Save changes to tasks.csv
6. Refresh the GUI table
7. End

### **Algorithm for Deleting a Task**

1. Start
2. User selects a task row in the GUI
3. Confirm deletion with a dialog box
4. If confirmed, remove the task from the ArrayList
5. Save changes to tasks.csv
6. Refresh the GUI table
7. End

### **Algorithm for Loading Tasks at Startup**

1. Start
2. Check if tasks.csv exists
3. If exists, read each line and parse task details
4. Create Task objects and insert into ArrayList
5. Populate the GUI table with these tasks
6. End

**Screenshots of the Code :**

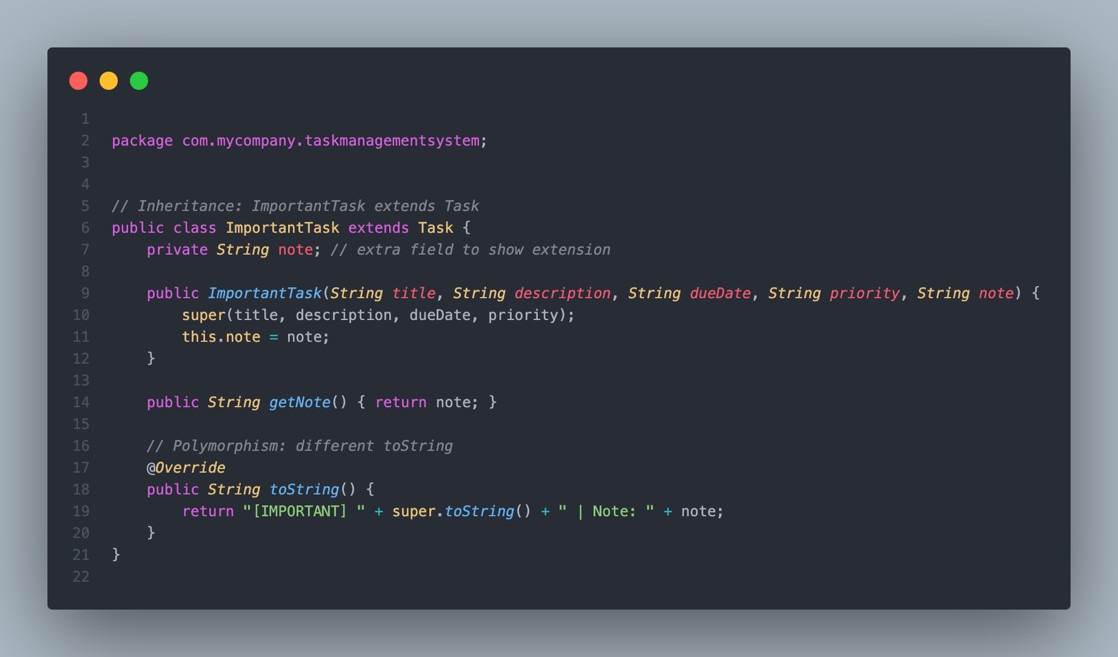
Implemented in **Java** with:

* Task.java → Encapsulation
* ImportantTask.java → Inheritance & Polymorphism
* TaskOperations.java → Abstraction (Interface)
* TaskManager.java → File Handling + Core CRUD logic
* TaskManagementSystem.java → GUI (Java Swing)

**Task.java (Encapsulation)**

****

**ImportantTask.java (Inheritance & Polymorphism)**

****

**TaskOperations.java (Abstraction with Interface)**

****

**TaskManager.java (File Handling & Data Storage)**

****

**TaskManager.java (File Handling & Data Storage)**

**TaskManagementSystem.java (Main GUI Implementation)**

****

**TaskManagementSystem.java (Main GUI Implementation)**

****

TaskManagementSystem.java (Main GUI Implementation)

****

**TaskManagementSystem.java (Main GUI Implementation)**

****

**Evidence of Core OOP Features Implemented:**

### **Encapsulation**

### Implemented in Task class using **private fields** and **getters/setters**.

|  |
| --- |
| private String title;  private String description;  private String dueDate;  public String getTitle() { return title; }  public void setTitle(String title) { this.title = title; } |

### **Inheritance**

### ImportantTask class inherits from Task.

|  |
| --- |
| class ImportantTask extends Task {  private String note;  public ImportantTask(String t, String d, String date, String p, String note) {  super(t, d, date, p);  this.note = note;  }  } |

**Polymorphism**

Overriding toString() method in both Task and ImportantTask.

|  |
| --- |
| @Override  public String toString() {  return "[IMPORTANT] " + super.toString() + " | Note: " + note;  } |

### **Abstraction**

### TaskOperations interface defines the **contract** for all task management operations.

|  |
| --- |
| interface TaskOperations {  void addTask(Task task);  void updateTask(int index, Task task);  void deleteTask(int index);  List<Task> getAllTasks();  } |

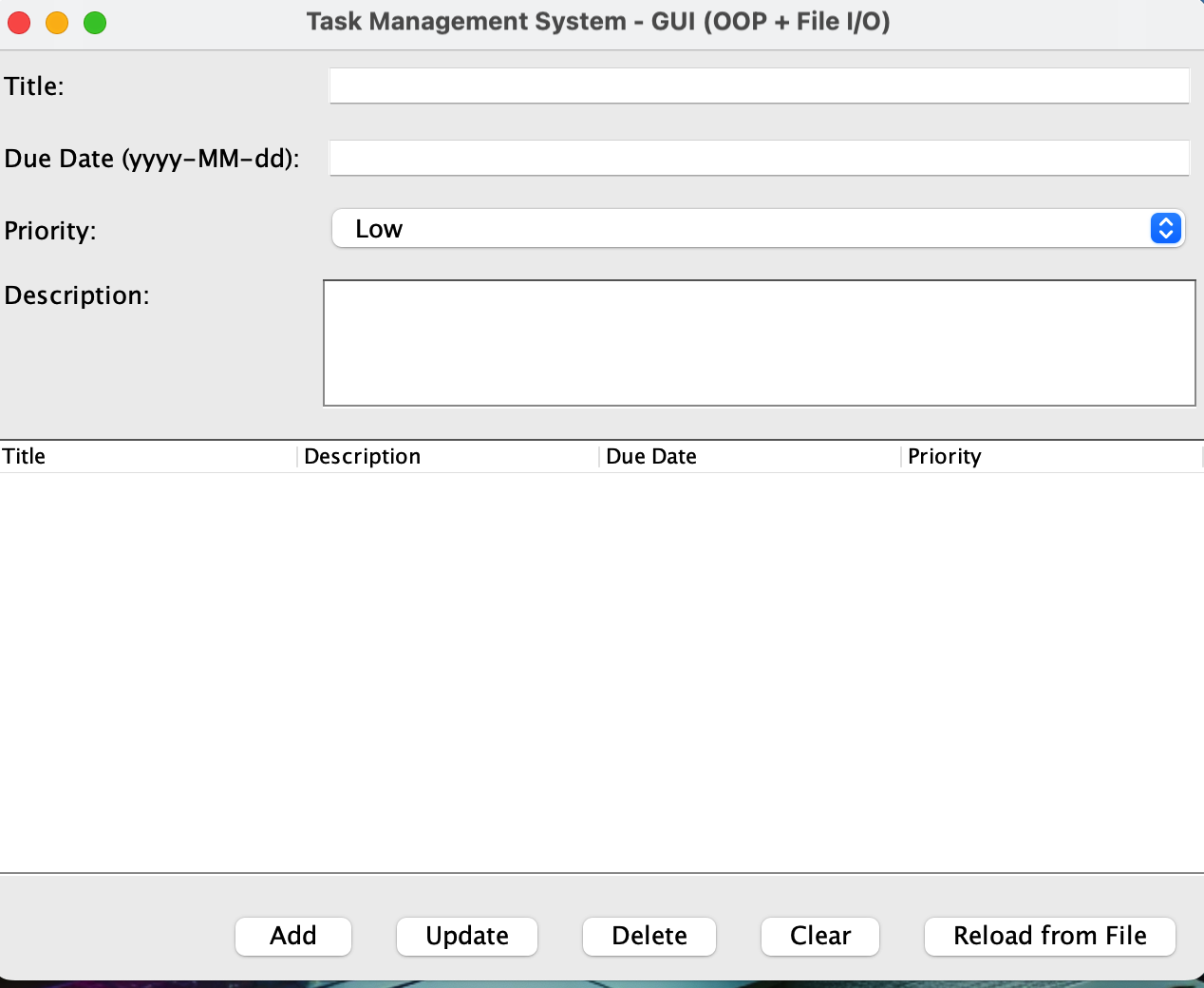
### **File Handling**

### Implemented in TaskManager class using **CSV-based save & load**.

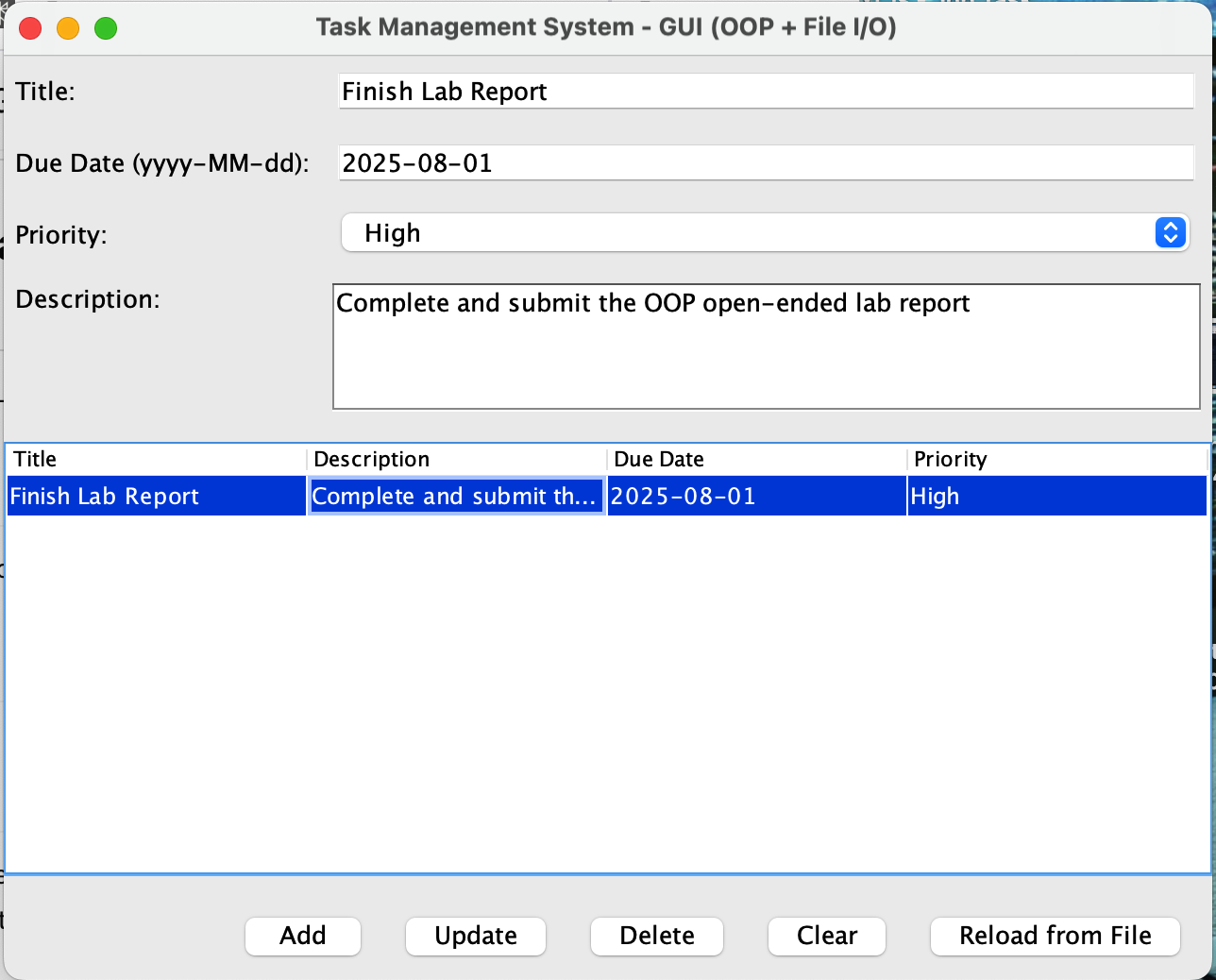
|  |
| --- |
| public void save() throws Exception {  try (BufferedWriter bw = Files.newBufferedWriter(file, StandardCharsets.UTF\_8)) {  bw.write("title,description,dueDate,priority\n");  for (Task t : tasks) {  bw.write(csv(t.getTitle()) + "," + csv(t.getDescription()) + "," +  csv(t.getDueDate()) + "," + csv(t.getPriority()) + "\n");  }  }  } |

**Screenshots of the GUI Implementation:**

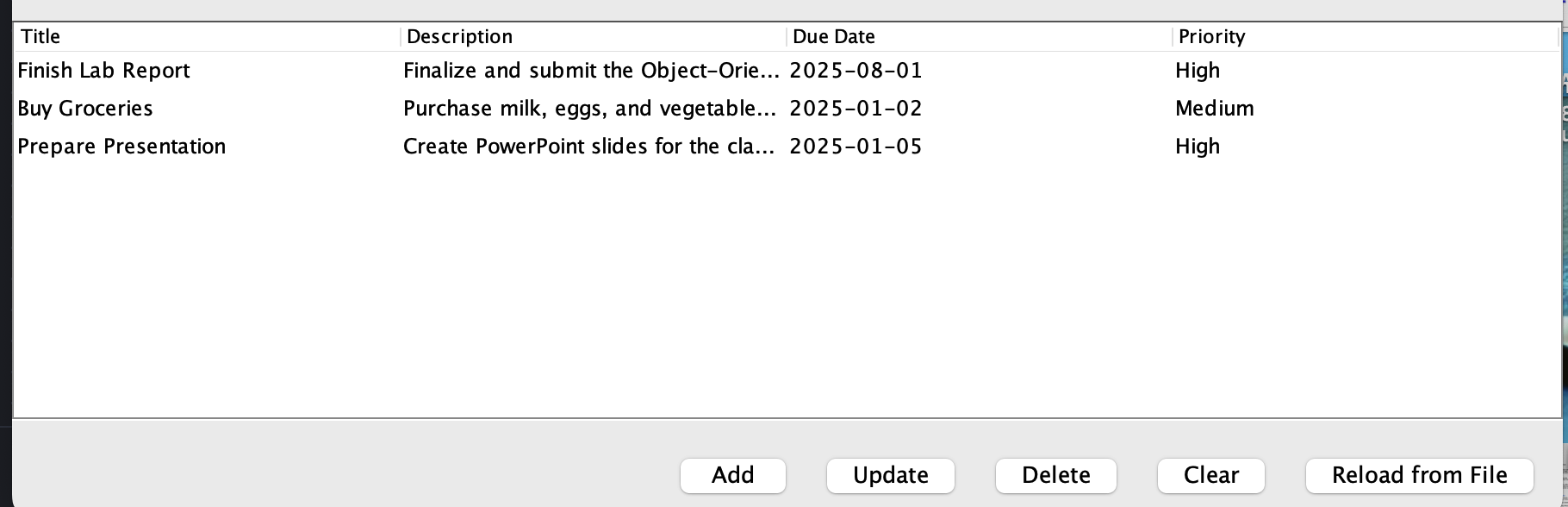
Insert screenshots taken while running the program:



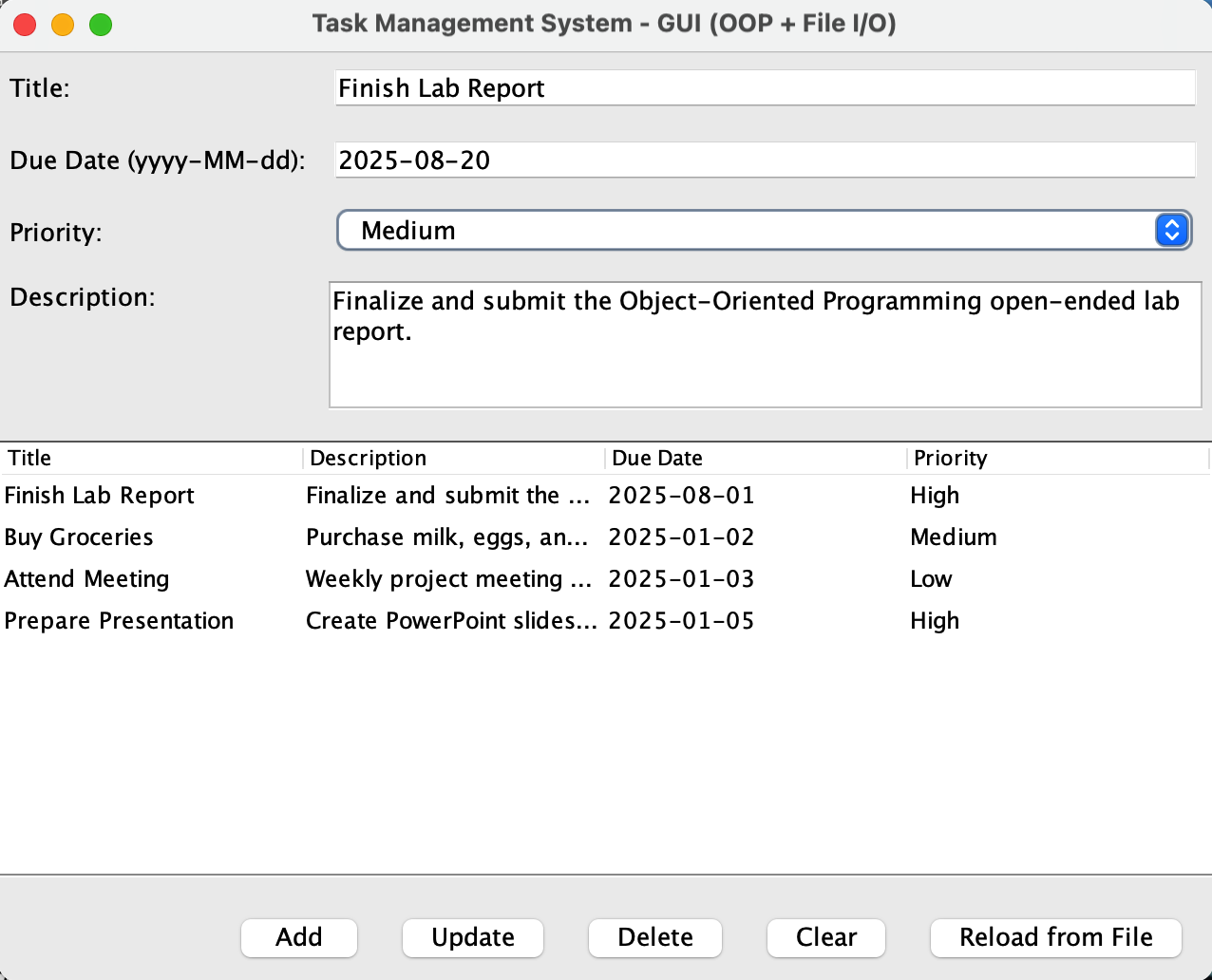
GUI Main Window (Empty Task List)



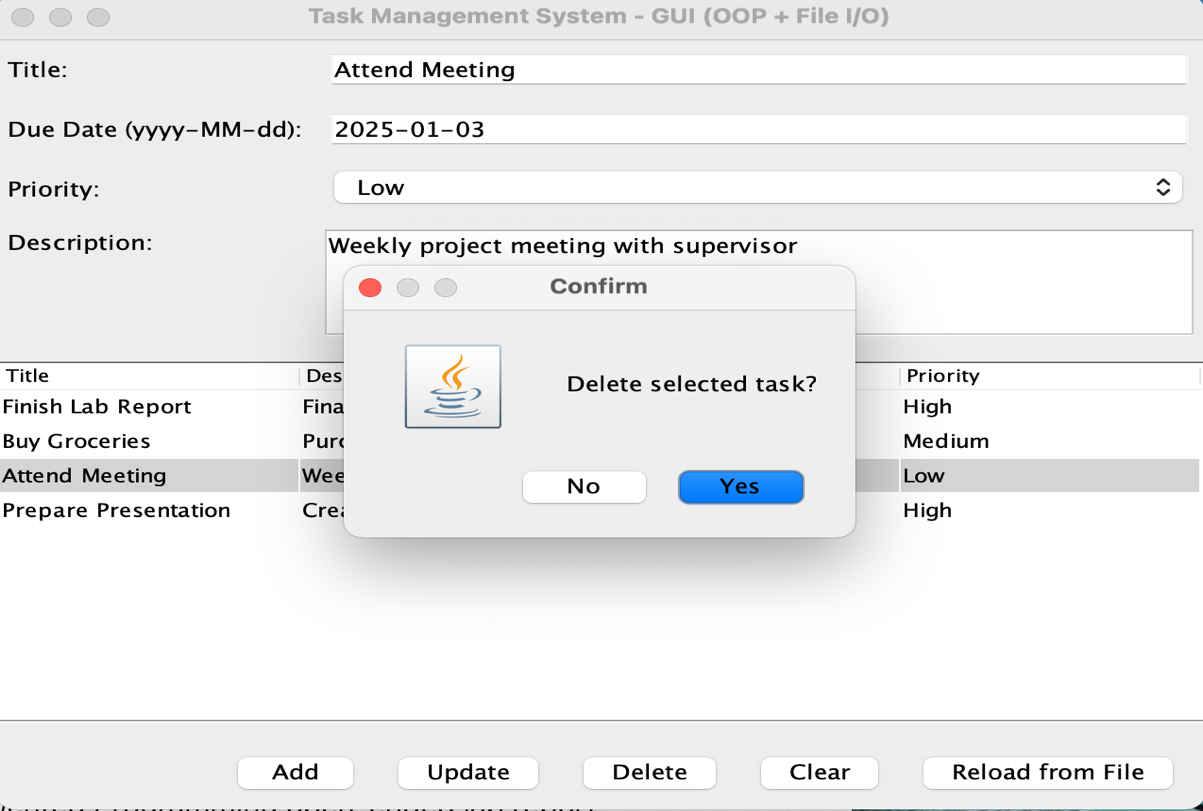
Adding a Task (Input Fields + Add Button)



Viewing Tasks in the Table



Updating a Task



Deleting a Task (Confirmation Dialog)

**Conclusion:**

This open-ended lab demonstrates a fully functional **Task Management System** built with **Java Swing, File Handling, and OOP principles**.  
The project successfully covers:

* **Encapsulation, Inheritance, Polymorphism, and Abstraction**
* **CRUD operations with persistent storage**
* **Modern GUI-based interface**

Thus, the software meets both the **given problem requirements** and the **open-ended features** (modern tools, GUI, OOP).