1. Introduction

The To-Do App is a task management application designed to allow users to manage tasks, mark them as completed, and organize them by due date. The application supports adding new tasks, viewing completed tasks, restoring completed tasks to the main list, and clearing all tasks from the completed tasks list.

This manual provides a detailed overview of the app's design and implementation. It covers the functionality of the key classes and explains how the components work together to achieve the app's goal.

---

2. System Overview

The app consists of three main components:

1. **Task Class**: Represents a single task, containing details such as name, date, priority, and completion status.
2. **TaskManager Class**: Manages the collection of tasks, providing functionality for adding, completing, retrieving, and deleting tasks.
3. **Main Class**: The primary entry point of the application. It manages the user interface and interacts with the `TaskManager` to update the task list and handle user actions.

The app uses JavaFX for the graphical user interface (GUI) to display tasks and interact with the user. The task list is divided into two categories: pending tasks (tasks that are yet to be completed) and completed tasks. Tasks are grouped by date, sorted by priority, and can be restored from the completed list back to the pending tasks.

---

3. Class Descriptions

3.1 Task Class

The `Task` class represents a single task in the application. It holds the task's name, due date, priority, and completion status.

- **Attributes**:
  - `String name`: The name or description of the task.
  - `LocalDate date`: The due date of the task.
  - `int priority`: The priority of the task, ranging from 1 (highest) to 5 (lowest).
  - `boolean completed`: Indicates whether the task is completed.
- **Methods**:
  - **Constructor**: Initializes a new task with the provided name, date, and priority.

- `String getName()`: Returns the task's name.
- `LocalDate getDate()`: Returns the task's due date.
- `int getPriority()`: Returns the task's priority.
- `boolean isCompleted()`: Returns whether the task is completed.
- `void setCompleted(boolean completed)`: Sets the task as completed or pending.
- `String toString()`: Returns a string representation of the task for display.

3.2 TaskManager Class

The `TaskManager` class is responsible for managing tasks, including adding tasks, completing tasks, and retrieving tasks for display. It stores tasks in two lists: one for pending tasks and another for completed tasks.

- **Attributes**:
  - `List<Task> pendingTasks`: A list that holds all pending tasks.
  - `List<Task> completedTasks`: A list that holds all completed tasks.
- **Methods**:
  - `void addTask(Task task)`: Adds a new task to the pending tasks list.
  - `void completeTask(Task task)`: Marks a task as completed and moves it from the pending list to the completed list.
  - `List<Task> getPendingTasks()`: Returns the list of all pending tasks.
  - `List<Task> getCompletedTasks()`: Returns the list of all completed tasks.
  - `void deleteTask(Task task)`: Deletes a task from either the pending or completed tasks list.

3.3 Main Class

The `Main` class is the entry point of the application. It creates the user interface and handles user actions like adding tasks, viewing completed tasks, and restoring tasks.

- **Attributes**:
  - `TaskManager taskManager`: An instance of the `TaskManager` class used to manage tasks.
  - `Stage primaryStage`: The primary stage (window) of the JavaFX application.
- **Methods**:
  - `void start(Stage primaryStage)`: The main method to launch the application. It initializes the primary stage and sets the main scene.
  - `Scene createMainTaskListScene()`: Creates the scene that displays the main task list, including buttons for adding tasks and viewing completed tasks.
  - `Scene createCompletedTasksScene()`: Creates the scene that displays completed tasks, with buttons to restore tasks and clear the completed tasks list.

    - `void updatePendingTasksListView(VBox container)`: Updates the list of pending tasks displayed in the GUI.
    - `void updateCompletedTasksListView(VBox container)`: Updates the list of completed tasks displayed in the GUI.
    - `void showAddTaskDialog(VBox pendingTasksContainer)`: Opens a dialog to allow the user to add a new task.
    - `void switchToCompletedTasksScene()`: Switches the view to the completed tasks scene.

---

4. User Interface Design

The app uses a simple and intuitive JavaFX interface with the following components:

1. **Main Task List Scene**:
    - **Buttons**:
        - `Add Task`: Opens a dialog for adding a new task.
        - `View Completed Tasks`: Navigates to the completed tasks view.
    - **Pending Tasks List**: Displays tasks grouped by date and sorted by priority. Each task has a checkbox to mark it as completed.
2. **Completed Tasks Scene**:
    - **Buttons**:
        - `Back to Main List`: Navigates back to the main task list.
        - `Clear All Tasks`: Deletes all tasks from the completed tasks list.
    - **Completed Tasks List**: Displays tasks that have been marked as completed, with a "Restore" button for each task to move it back to the main list.

---

5. Data Flow and Task Management

- **Adding Tasks**: When the user adds a new task, it is added to the `pendingTasks` list of the `TaskManager` class. The task is immediately displayed in the GUI.
- **Marking Tasks as Completed**: When the user checks a task's checkbox, the task is moved from the `pendingTasks` list to the `completedTasks` list. The GUI is then updated to reflect this change.
- **Restoring Completed Tasks**: If a user wants to restore a completed task back to the main task list, they can click the "Restore" button next to the task on the completed tasks screen. This moves the task back to the `pendingTasks` list.
- **Clearing Completed Tasks**: When the user clicks the "Clear All Tasks" button, all tasks in the `completedTasks` list are removed, and the completed tasks view is updated.

---

6. Class Relationships

1. **Task**:
    - Each `Task` object contains information about a single task.
    - A `Task` is managed by the `TaskManager` class, which handles operations like adding, completing, and deleting tasks.
2. **TaskManager**:
    - The `TaskManager` class manages multiple `Task` objects.
    - It provides methods for adding tasks, marking tasks as completed, retrieving pending and completed tasks, and deleting tasks.
    - It is used by the `Main` class to manage the task data and update the user interface.
3. **Main**:
    - The `Main` class is responsible for creating and displaying the user interface.
    - It uses the `TaskManager` class to retrieve tasks and update the views accordingly.

---

7. Future Enhancements

- **Task Editing**: Allow users to edit existing tasks, such as changing the task name, date, or priority.
- **Task Filtering**: Implement filters to show tasks based on priority, completion status, or due date.
- **Persistence**: Implement file storage to save tasks between sessions, ensuring that tasks are not lost when the application is closed.