CSE2006 - Programming in Java

Project Report: Persistent Task Manager (PTM)

Name: Kirti Yogi **Registration no. : 24MIM10188**

1.Introduction

The primary objective of the Persistent Task Manager (PTM) is to develop a robust, console-based application in Java that allows users to effectively manage their daily and long-term tasks. The core focus is on implementing data persistence to ensure that tasks are saved across program executions, providing a reliable utility for task tracking.

2. System Analysis and Design

2.1. Technology Stack

The Persistent Task Manager is built upon a standard Java platform, augmented by specific Input/Output (I/O) and utility classes to achieve core functionality. The system's design is centered around a dual-persistence approach, requiring specific technology choices to handle both binary object serialization and human-readable data export.

- Programming Language: Java (JDK 8+) is selected for its strong object-oriented paradigm, platform independence, and robust, built-in libraries necessary for handling file persistence and modern date/time complexities.
- Primary Persistence: Java Serialization (tasks.dat) is utilized for its efficiency in saving and loading the complete object graph (the List<Task> object) for the application's internal use.
- Secondary Persistence: CSV File I/O (tasks.csv) is implemented as a crucial backup and inspection layer, providing a simple, human-readable format for auditing saved data.
- Date/Time Handling: java.time.LocalDate is used to manage due dates effectively and perform reliable date comparisons for sorting.

2.2 Functional Requirements

The system is designed to meet the core functional requirements necessary for a robust task management application:

- Task Creation: The system must allow the user to input and create a new task, requiring a Title, Description, Priority (1-3), and a validated Due Date in the dd-MM-yyyy format.

- Task Viewing (Read): Users must be able to view all tasks currently loaded in the system, displaying their details and current completion status.
- Task Update: The system must provide a mechanism to mark an existing task as completed via user selection.
- Task Deletion: Users must be able to permanently remove a task by index.
- Sorting: The application must support sorting the task list based on two distinct criteria: Priority (High to Low) and Due Date (Closest First).
- Data Persistence: Task data must be automatically saved after every modifying operation (Add, Mark Complete, Delete) and upon program exit, ensuring continuous state integrity.

2.3 System Components (Classes)

The application's architecture adheres to modular design principles, specifically the Single Responsibility Principle (SRP), by separating concerns across three highly cohesive classes. This structure ensures maintainability and scalability.

- Task.java (Model): This class defines the data structure and core state (title, priority, completion status) of a single task item. It is the only class that implements the Serializable interface to enable primary persistence.
- TaskManager.java (Controller/Logic): This class manages the collection of Task objects, implements the dual persistence logic (DAT and CSV), and contains all sorting and CRUD operation logic.
- Main.java (View/Interface): This class contains the application's entry point (main method), manages the console-based menu loop, processes user choices, and enforces rigorous input validation for critical fields like date and priority.

3. Implementation and Algorithms

3.1 Task Management Operations (CRUD)

The TaskManager.java class serves as the core logic layer, handling the Create, Read, Update, and Delete (**CRUD**) operations on the internal List<Task> collection. After any state-changing operation (Add, Mark Complete, Delete), the manager automatically invokes saveTasks() to ensure immediate data persistence.

- **Create (Add Task):** The Main.java interface collects all task parameters and delegates the task creation to manager.addTask(). This method validates inputs (Title, Description) and, upon successful creation of a Task object, appends it to the internal list.

- **Update (Mark Complete):** The manager.markTaskCompleted(index) method retrieves the specified task and calls its **mutator method**, Task.markCompleted(), which sets the boolean status flag isCompleted = true.
- **Delete:** The manager.deleteTask(index) method uses the List.remove(index) functionality to permanently excise the task from the collection, maintaining data integrity by immediately invoking the save operation.

3.2 Dual Persistence Mechanism

The implementation of a **dual persistence strategy** is critical to the system's reliability and integrity. This mechanism is housed entirely within the TaskManager.java class:

- **Primary Persistence (Serialization):** An ObjectOutputStream is used to write the entire tasks list (a complex Java object graph) to the binary file tasks.dat. This is the application's most efficient and reliable method for saving and loading the complete state.
- **Secondary Persistence (CSV Export):** Immediately following the primary save, the saveTasksToCsv() method writes a simplified, comma-separated version of all task data to tasks.csv. This plain-text file serves as a **human-readable audit log and backup**.

3.3 Custom Sorting Algorithm

The TaskManager.getSortedTasks(String sortBy) method implements two distinct, efficient sorting algorithms using Java 8's functional features. This approach leverages **Comparator chaining** to handle multi-criteria sorting, analogous to advanced query optimization in data retrieval.

- **Priority Sort:** Tasks are sorted primarily by **Priority** (1 to 3) and secondarily by **Completion Status** (isCompleted) to group pending tasks of the same priority together.
- **Date Sort:** Tasks are sorted primarily by **Due Date** (earliest first) and secondarily by **Completion Status**.

4. Conclusion

The Persistent Task Manager (PTM) is a successful implementation of a console utility that effectively manages task data. The core challenge of data persistence was solved using a reliable dual-saving

mechanism. The system demonstrates a high degree of modularity and robustness through clear class separation and rigorous input validation.

5. Future Scope

Future work could explore:

1. **Task Editing (Hybrid Approach):** Implement an explicit "Edit Task" option to modify existing fields (Title, Description, Date) without requiring the user to delete and re-add the task.
2. **Advanced Filtering:** Add dynamic filtering capabilities, such as showing only tasks due today, or only pending tasks, to improve user efficiency.
3. **Cross-platform Data Storage:** Replace file-based serialization with a small, embedded database (e.g., SQLite) to handle more complex querying and larger datasets.
4. **Graphical User Interface (GUI):** Migrate the current console interface to a modern desktop application using JavaFX or Swing.