CLI Task Manager Application

Language: Java
Date: 12/09/2023
Developer: Pierre Harbin

Plan for User Interface

Main Menu:

Add Task:

Description: Allows the user to add a new task to the task manager.
Use Case: The user enters a task description when prompted, and the system adds the task to the task manager. After adding, the system may apply decorators (e.g., PriorityDecorator) to the task.

Display Tasks:

Description: Displays a list of all tasks in the task manager.
Use Case: The user selects this option to view the list of tasks. The system prints out the task IDs, descriptions, and statuses. This helps the user keep track of their tasks.

Mark Task as Complete:

Description: Allows the user to mark a specific task as complete.
Use Case: The user provides the task ID to mark as complete. The system updates the task's status, and the user receives confirmation. This is useful for tracking task progress.

Display Users:

Description: Displays a list of system users.
Use Case: The user selects this option to view the list of users. The system prints out the usernames. This feature might be useful in scenarios where task assignments or ownership are relevant.

Exit:

Description: Exits the application.
Use Case: The user selects this option to exit the program gracefully.

Additional Notes:

User Creation:

Description: Users are created during the initialization of the program. However, in a real-world scenario, there could be an option to manage users, allowing the addition, deletion, or modification of user details.

Decorator Implementation:

Description: Decorators, like PriorityDecorator, are applied to tasks to add additional information or features. In this case, the PriorityDecorator adds priority information to tasks. Future decorators could extend this functionality.

Looping Structure:

Description: The program uses an infinite loop to repeatedly display the main menu until the user chooses to exit. This ensures a continuous interaction flow.

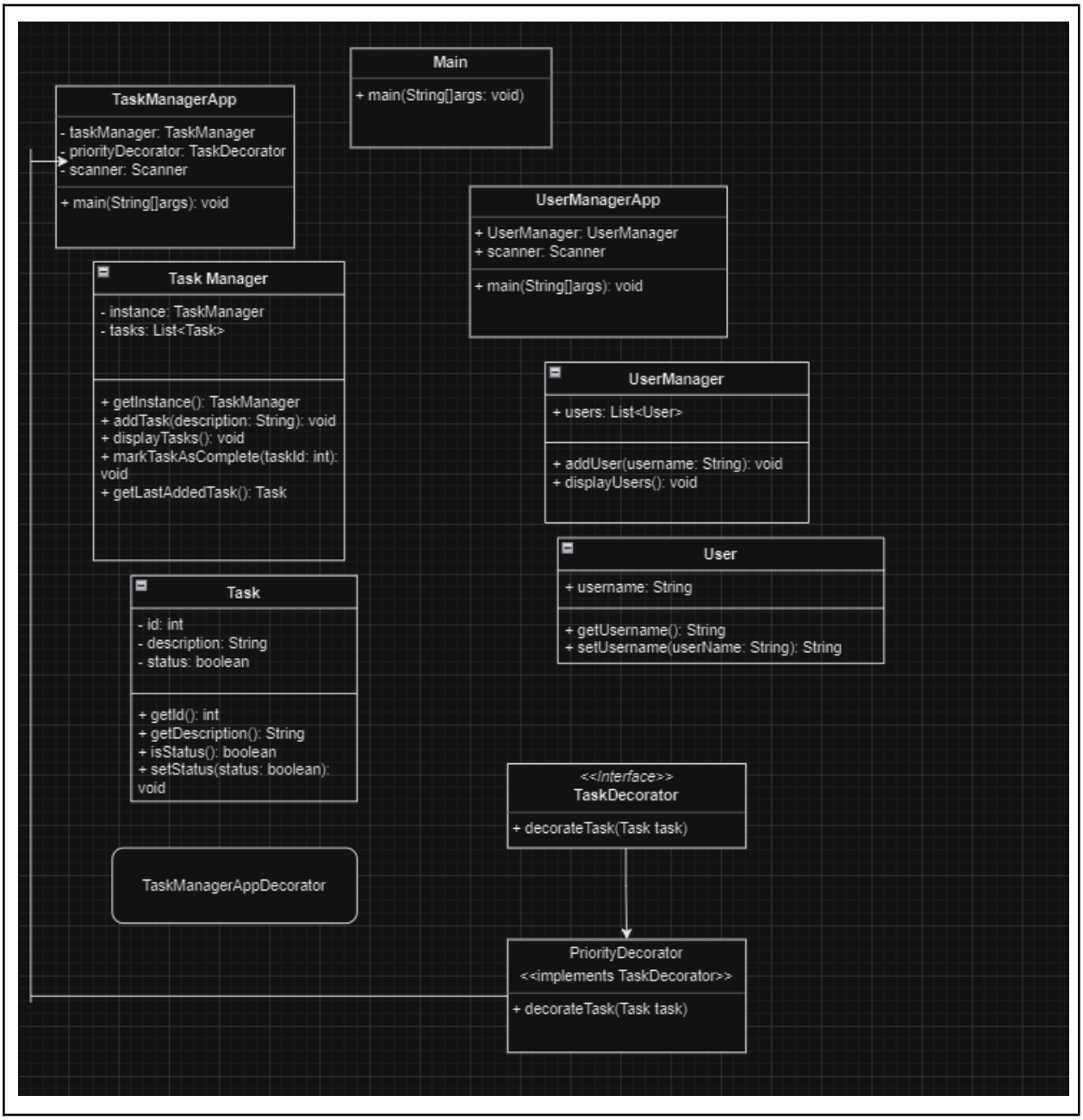
Error Handling:

Description: The program may include error handling to manage scenarios like invalid input, task not found, or other exceptional cases. Users would receive appropriate messages guiding them on how to correct or address the issues.

User Interaction:

The user interacts with the system by selecting numeric options from the menu.
Input validation is implemented to handle invalid inputs.
The system provides feedback to the user after each action, confirming successful task addition, marking tasks as complete, etc.

Project Diagram:



Java Source Code

Github: [Task Manager Repository]

In order to run the following:

First Compile the Program	**Run the Main Class**

javac Task.java TaskManager.java TaskDecorator.java PriorityDecorator.java \
  TaskManagerApp.java TaskManagerAppDecorator.java User.java UserManager.java \
  UserManagerApp.java UserTaskManagerApp.java
	java UserTaskManagerApp

Summary

The project aimed to create a console-based user and task management application in Java, incorporating design patterns such as the Singleton, Factory, and Decorator patterns. The initial plan involved designing classes for tasks (Task, TaskManager, TaskDecorator, PriorityDecorator, TaskManagerApp, and TaskManagerAppDecorator), users (User and UserManager), and a combined application for managing both users and tasks (UserTaskManagerApp). The design also included a menu-based interaction system for users to add tasks, mark tasks as complete, manage users, and set decorators.

During the implementation, the design evolved to ensure better encapsulation and separation of concerns. The Singleton pattern was applied to the TaskManager class to ensure a single instance throughout the program, and the Decorator pattern was used to dynamically add behavior to tasks. The PriorityDecorator class was introduced to simulate the addition of priority information to tasks. The code was structured to allow users to switch between task and user management seamlessly.

The implementation showcased the flexibility of the design patterns, allowing for easy extension and modification. However, some deviations occurred during implementation, such as simplifying the decorator application by applying a default priority decorator to each added task. Additionally, the initial plan did not include the Factory pattern, which could be considered for future enhancements.

If continuing to work on this project, improvements could be made by introducing error handling for user input, enhancing the decorator pattern with more sophisticated decorators, and potentially incorporating a database for persistent data storage. Further modularization and unit testing could also contribute to a more robust and maintainable codebase. Overall, the project demonstrated the application of design patterns in a real-world scenario and provided insights into the iterative nature of software development.