**Sabina Tleugabyl**

**SE-2305**

**Project Description**

**Task Manager Application** is a Java-based application designed to help users manage tasks efficiently. The application uses several design patterns, including the Factory, Strategy, Observer, and Facade patterns, to structure the project in a way that is modular, maintainable, and easy to extend. This application allows users to add tasks, set their priorities, and change the status of the tasks as they move from "Pending" to "In Progress" and "Completed". The goal of the project is to provide a straightforward and flexible tool for managing tasks in a clear and organized way.

**Project Goal**

The main goal of this project is to develop a task management application where users can:

1. Add Tasks: Users can create new tasks by entering their title, description, and priority level.
2. Update Task Status: The status of tasks can be updated to reflect whether they are in progress or completed.
3. Display Tasks: Tasks are displayed in the user interface with their respective titles, priorities, and statuses.
4. Efficient Task Management: Users can organize and track the progress of their tasks based on priority and status.

**Features**

* Task Creation: The application allows users to create tasks by entering details such as task name, description, and priority.
* Task Management: Tasks can be updated with a status of "In Progress" or "Completed".
* Task List: The application displays a list of tasks, providing users with an overview of all their tasks.
* Priority Assignment: Users can assign different priorities to tasks such as "Low", "Medium", and "High".
* Design Patterns Implementation: The use of design patterns such as Factory, Strategy, Observer, and Facade ensures that the application is well-structured and flexible for future expansions or changes.

**Target Audience**

This task manager is aimed at users who need a simple, user-friendly tool to organize their tasks, whether it's for personal use or small-scale team collaboration. The application is particularly useful for professionals, students, or anyone managing multiple tasks and projects simultaneously.

**Project Architecture**

The Task Manager project follows the MVC (Model-View-Controller) design pattern, ensuring that the application is well-organized and adheres to a clear separation of concerns. This pattern helps in making the application scalable, maintainable, and easier to test.

* **Model:** Contains the core logic of the application, including task data, task status management, and task strategies. Classes like Task, TaskManager, TaskStatus, and task strategy implementations (e.g., TaskCompletedStrategy, TaskInProgressStrategy) belong to the model layer.
* **View:** Manages the user interface. It displays tasks and provides forms for task creation and status updates. The view listens for user input and reflects changes in the model. Classes such as TaskView and TaskDisplay handle the presentation layer.
* **Controller:** Acts as the intermediary between the model and the view. The controller responds to user actions, updates the model, and notifies the view. In this project, the TaskController and TaskFacade are responsible for handling user requests and interacting with the model.

**Design Patterns Used**

Several design patterns were implemented in this project to address specific software design challenges, providing flexibility, maintainability, and separation of concerns. Below is an overview of the patterns used and their roles in the system.

**1. Singleton Pattern (TaskManager)**

* **Purpose**: Ensures that only one instance of the TaskManager class exists at any given time.
* **Role in the system**: The TaskManager class is responsible for managing tasks. The Singleton Pattern ensures that no multiple instances of the task manager are created, preventing inconsistencies when tasks are managed. This is particularly important for maintaining a central point of control over the task data.

**2. Factory Pattern (TaskFactory)**

* **Purpose**: Creates Task objects based on parameters like title, description, and priority.
* **Role in the system**: The TaskFactory abstracts the creation of Task objects, allowing the application to easily instantiate tasks with different properties. It simplifies object creation by hiding the complexity from other parts of the code, making it easier to introduce new types of tasks in the future.

**3. Strategy Pattern (TaskStrategy)**

* **Purpose**: Defines a family of algorithms for task status changes and allows them to be interchangeable.
* **Role in the system**: The TaskStrategy interface, along with its concrete implementations (e.g., TaskCompletedStrategy, TaskInProgressStrategy), defines how tasks transition between different statuses (e.g., from "Pending" to "In Progress" or "Completed"). This allows the task status to change dynamically based on different conditions without changing the core Task class.

**4. Observer Pattern (TaskObserver and TaskStatusObserver)**

* **Purpose**: Allows an object (the observer) to be notified of changes in another object (the subject) without tightly coupling the two.
* **Role in the system**: The TaskObserver interface and its implementation TaskStatusObserver enable real-time updates when the status of a task changes. The observer pattern ensures that different components, such as the UI, are updated automatically when the task status changes, without the need for manual intervention. This decouples the Task class from its viewers, improving flexibility and scalability.

**5. Facade Pattern (TaskFacade)**

* **Purpose**: Provides a simplified interface to a complex subsystem, hiding its complexity.
* **Role in the system**: The TaskFacade class offers a simplified interface for managing tasks, encapsulating the complexity of interacting with the Task, TaskManager, and other related classes. It provides a single point of access for adding tasks, fetching task data, and modifying task states, which simplifies the interaction for the View and Controller components.

**6. Adapter Pattern (TaskAdapter)**

* **Purpose**: Converts one interface to another that the client expects, enabling incompatible interfaces to work together.
* **Role in the system**: The TaskAdapter class acts as a bridge between the task data (Model) and the graphical user interface (View). It transforms the complex Task object into a simplified format that can be displayed by JavaFX components, such as ListView or ComboBox. The adapter decouples the View from the Task model, making it easier to update the UI components without changing the core business logic. This allows the system to adapt to different UI components or models without affecting the overall structure of the application.

**UML Diagrams**

**com.taskmanager**

**├── adapter**

**│ └── TaskAdapter.java**

**├── controller**

**│ ├── TaskController.java**

**│ └── TaskFacade.java**

**├── model**

**│ ├── HighPriorityTask.java**

**│ ├── Task.java**

**│ ├── TaskCompletedStrategy.java**

**│ ├── TaskFactory.java**

**│ ├── TaskInProgressStrategy.java**

**│ ├── TaskManager.java**

**│ ├── TaskObserver.java**

**│ ├── TaskStatus.java**

**│ ├── TaskStatusObserver.java**

**│ └── TaskStrategy.java**

**├── view**

**│ ├── TaskDisplay.java**

**│ └── TaskView.java**

**└── Main.java**

**Technologies Used**

* Java: The application is built using Java, leveraging object-oriented principles.
* JavaFX: JavaFX is used for creating the graphical user interface (GUI), allowing for a rich and interactive user experience.
* Design Patterns: The use of Factory, Strategy, Observer, and Facade design patterns ensures that the code is modular, flexible, and scalable.

**Limitations**

* Persistence: The application currently stores tasks in memory. If the application is closed, all tasks are lost. To address this, a persistent storage solution like a database or file system could be integrated.
* Multi-User Support: The system is designed for single-user use. To support multiple users or teams, additional features such as user authentication and task sharing could be added.
* Scalability: As the number of tasks grows, the performance of the application might be impacted if a more efficient data storage solution is not implemented.

**Future Improvements**

1. Database Integration: Integrating a database (such as MySQL or SQLite) to persist tasks between sessions.
2. User Authentication: Adding support for multiple users with login and registration functionality.
3. Task Filtering and Sorting: Implementing filtering and sorting mechanisms to help users find tasks based on priority or status.
4. Mobile Version: Developing a mobile version of the task manager for easier task management on the go.

**Conclusion**

The Task Manager application is a robust task management tool built with the principles of object-oriented programming and design patterns to ensure flexibility, maintainability, and scalability. The use of JavaFX for the user interface and the implementation of core design patterns makes the application not only functional but also well-structured for future extensions. This project can be used by individuals or teams to organize and track tasks efficiently.