**Class A Application (1-5 classes): Personal Task Manager**

**Experiment 1 (Small Application, Low Detail initial Prompt) Class A Application (1-5 classes): Personal Task Manager**

**Description:** A simple application that allows users to manage their daily tasks and to-dos.

**1. Background:** The Personal Task Manager application is envisioned as a streamlined tool for individual users to manage their daily tasks and to-dos efficiently. The application will provide a user-friendly interface for adding, updating, and tracking tasks, with features tailored to enhance personal productivity. It will be developed in Java, leveraging its portability and robust ecosystem to ensure a reliable and cross-platform solution.

**2. Basic Application Design:**

* **Main Menu:** Options to view tasks, add a new task, delete a task, and mark a task as completed.
* **Add Task:** Input fields for task name, description, due date, and reminder time.
* **View Tasks:** Displays a list of tasks with their due dates. Tasks close to their due date or overdue are highlighted.

**3. Database Structure:**

* A lightweight MySQL database will be used to store user tasks, due to its simplicity and ease of integration with Java applications.
* The database schema will include tables for tasks, with fields for task ID, title, description, priority, deadline, and completion status.
* Relationships will be straightforward, focusing on a single user's tasks without the need for user account management in this version.

**4. GUI Framework:**

* The Java Swing framework will be utilized to create the graphical user interface, known for its robustness and ease of use.
* The interface will be designed to be intuitive, with a focus on minimalism and clarity to avoid overwhelming the user.
* Common UI components like buttons, text fields, checkboxes, and date pickers will be used to ensure a familiar user experience.

**5. Deployment:**

* The application will be packaged as a Java Archive (JAR) file for easy distribution and execution on any platform with a compatible Java Runtime Environment (JRE).
* Deployment instructions will be provided to guide users through the installation process on different operating systems.

**Experiment 2 (Small Application, Medium Detail initial Prompt) Class A Application (1-5 classes): Personal Task Manager**

**Description:** A simple application that allows users to manage their daily tasks and to-dos.

1. **Background:** The Personal Task Manager application is envisioned as a streamlined tool for individual users to manage their daily tasks and to-dos efficiently. The application will provide a user-friendly interface for adding, updating, and tracking tasks, with features tailored to enhance personal productivity. It will be developed in Java, leveraging its portability and robust ecosystem to ensure a reliable and cross-platform solution.
2. **Basic Application Design:**
3. MainApplication: This primary class initiates the application, presenting users with an intuitive interface that displays daily tasks. It includes navigation capabilities to peruse tasks across different dates, enhancing user engagement with temporal task management.
4. TaskManagerUI: TaskManagerUI is tasked with the graphical user interface, offering functionalities for task addition, modification, and completion. It allows users to input task details such as title, description, priority, and deadline. A search feature is incorporated to expedite the retrieval of tasks, while a summary view contrasts pending and completed tasks, providing a visual representation of productivity.
5. Task: The Task class encapsulates the data model for individual tasks, defining attributes such as title, description, priority, deadline, and completion status, thereby structuring the task information in a coherent and accessible format.
6. TaskController: Serving as the intermediary, the TaskController class manages the interaction between the TaskManagerUI and the Task data model. It processes user actions, invokes data persistence methods, and ensures that the user interface reflects the current state of tasks.
7. TaskDatabase: The TaskDatabase class is responsible for persistent storage operations. It interfaces with the underlying database to facilitate the creation, retrieval, update, and deletion of task records, ensuring data integrity and consistency.
8. **Modular Program Design:** Each function is developed as a separate Java class, ensuring maintainability and scalability.
9. **Database Structure:**
10. **Tasks Table**: This is the central table of the database, where each task is recorded. The table includes columns for **TaskID** (a unique identifier for each task), **Title**, **Description**, **Priority** (which may be an enumeration of values like Low, Medium, High), **Deadline** (a date), and **Status** (to indicate if the task is pending, in progress, or completed).
11. **Users Table**: Although not explicitly mentioned in the application design, a Users table can be included to manage multiple users if needed. It would contain **UserID**, **Username**, **PasswordHash**, and other relevant user information.
12. **Task-User Relationship**: If the application supports multiple users, a relationship between the Tasks and Users tables is necessary. This can be a one-to-many relationship where one user can have many tasks, but each task is associated with only one user.
13. **Indices**: To optimize the search functionality, indices on the **Title**, **Priority**, and **Deadline** columns of the Tasks table will be created. This will expedite query execution, especially when filtering and sorting tasks based on these attributes.
14. **Normalization**: The database will be normalized to at least the third normal form (3NF) to eliminate data redundancy and ensure data integrity. This means that every non-primary attribute is non-transitively dependent on the primary key in each table.
15. **GUI Framework:** Utilizes Java's Swing framework, ensuring a consistent and responsive user experience.
16. **Error Handling:** Basic error messages for common issues like "Task Not Found" or "Invalid Input".
17. **Authentication:** Given the internal nature of the application, no authentication layer is required.
18. **Concurrency:** Designed for individual use, ensuring data integrity without the need for concurrent access handling.
19. **External Libraries:** While the core functionality relies on Java's standard libraries, external libraries can be integrated for enhanced database connectivity or specialized GUI components.
20. **Deployment:** Packaged as a standalone application, it's deployable on both local VMWare Virtual Machines or cloud platforms for broader accessibility.

**Experiment 3 (Small Application, High Detail initial Prompt) Class A Application (1-5 classes): Personal Task Manager**

**Description:** A simple application that allows users to manage their daily tasks and to-dos.

1. **Background:** The Personal Task Manager application is envisioned as a streamlined tool for individual users to manage their daily tasks and to-dos efficiently. The application will provide a user-friendly interface for adding, updating, and tracking tasks, with features tailored to enhance personal productivity. It will be developed in Java, leveraging its portability and robust ecosystem to ensure a reliable and cross-platform solution.
2. **Basic Application Design:**
3. MainApplication: This primary class initiates the application, presenting users with an intuitive interface that displays daily tasks. It includes navigation capabilities to peruse tasks across different dates, enhancing user engagement with temporal task management.
4. TaskManagerUI: TaskManagerUI is tasked with the graphical user interface, offering functionalities for task addition, modification, and completion. It allows users to input task details such as title, description, priority, and deadline. A search feature is incorporated to expedite the retrieval of tasks, while a summary view contrasts pending and completed tasks, providing a visual representation of productivity.
5. Task: The Task class encapsulates the data model for individual tasks, defining attributes such as title, description, priority, deadline, and completion status, thereby structuring the task information in a coherent and accessible format.
6. TaskController: Serving as the intermediary, the TaskController class manages the interaction between the TaskManagerUI and the Task data model. It processes user actions, invokes data persistence methods, and ensures that the user interface reflects the current state of tasks.
7. TaskDatabase: The TaskDatabase class is responsible for persistent storage operations. It interfaces with the underlying database to facilitate the creation, retrieval, update, and deletion of task records, ensuring data integrity and consistency.
8. **Modular Program Design**: Each function (add, update, delete, report) is developed as a separate module, ensuring maintainability and scalability. When generating Java classes, please generate the complete code.
9. **Database Structure**:

Database Structure Details: The database for the Personal Task Manager application is designed to be robust and efficient, ensuring quick access and secure storage of task data. The structure is as follows:

a. **Tasks Table**: This is the central table of the database, where each task is recorded. The table includes columns for **TaskID** (a unique identifier for each task), **Title**, **Description**, **Priority** (which may be an enumeration of values like Low, Medium, High), **Deadline** (a date), and **Status** (to indicate if the task is pending, in progress, or completed).

b. **Users Table**: Although not explicitly mentioned in the application design, a Users table can be included to manage multiple users if needed. It would contain **UserID**, **Username**, **PasswordHash**, and other relevant user information.

c. **Task-User Relationship**: If the application supports multiple users, a relationship between the Tasks and Users tables is necessary. This can be a one-to-many relationship where one user can have many tasks, but each task is associated with only one user.

d. **Indices**: To optimize the search functionality, indices on the **Title**, **Priority**, and **Deadline** columns of the Tasks table will be created. This will expedite query execution, especially when filtering and sorting tasks based on these attributes.

e. **Normalization**: The database will be normalized to at least the third normal form (3NF) to eliminate data redundancy and ensure data integrity. This means that every non-primary attribute is non-transitively dependent on the primary key in each table.

f. **Foreign Keys**: If the Users table is included, **UserID** will be a foreign key in the Tasks table to establish the relationship between users and their tasks.

g. **Backup and Recovery**: The database will include an automated backup system to prevent data loss. Recovery procedures will also be in place to restore data to a consistent state in case of a system failure.

h. **Scalability**: The database design will accommodate future scalability needs, allowing for additional tables or columns as the application grows in complexity and functionality.

i. No other tables are needed for this application

1. **GUI Framework**: Utilizes Java's Swing framework, ensuring a consistent and responsive user experience. No specific branding is needed.
2. **Error Handling**: Basic error messages for common issues like "Task Not Found" or "Invalid Input". No other guidance is needed.
3. **Authentication**: Given the internal nature of the application, no authentication layer is required.
4. **Concurrency**: Designed for individual use, ensuring data integrity without the need for concurrent access handling.
5. **External Libraries**: While the core functionality relies on Java's standard libraries, external libraries can be integrated for enhanced database connectivity or specialized GUI components. NO external 3rd party libraries should be used stick to standard Java libraries.
6. **Deployment**: Packaged as a standalone application, it's deployable on both local VMWare Virtual Machines or cloud platforms for broader accessibility.
7. **User Roles and Permissions:** Single user application.
8. **Data Validation and Constraints:** Validate the status (pending, in-progress, complete) and priority (low, medium, high) fields. The date field should be numeric.
9. **Backup and Recovery:** Not applicable
10. **Performance Requirements:** No specific requirements.
11. **Integration with Other Systems:** Not applicable.
12. **User Experience (UX) and User Interface (UI) Design:** No specific design guidelines, color schemes, or branding elements are provide – use your judgement.
13. **Logging and Auditing:** Not applicable.
14. **Feedback Mechanism:** Not applicable.
15. **Training and Documentation:** Consideration for user training materials or documentation to help faculty and staff get acquainted with the application. Develop after application is generated.
16. **Scalability and Future Expansion:** Not applicable.