ToDoPro Task Manager Program

1. Problem Analysis and Algorithm Design

Problem Analysis

The program addresses the need for a simple, user-friendly task management system that allows users to:

- Add new tasks
- Mark tasks as completed
- Delete tasks
- Clear completed tasks
- Save tasks to a file
- Load tasks from a file
- Exit the program with save confirmation

Algorithm Design

The program follows an object-oriented design with these key components:

1. **ToDo Class**: Represents individual tasks with properties like description, completion status, and timestamps.
2. **ToDoProManager Class**: Manages the collection of tasks and handles all business logic.
3. **NewJFrame Class**: Provides the graphical user interface (GUI) using Java Swing.

The main workflow:

- User interacts with GUI components (buttons, text fields)
- Event handlers in NewJFrame call appropriate methods in ToDoProManager
- ToDoProManager updates its internal task list and notifies the GUI to refresh
- For file operations, the manager uses Java serialization to save/load task data

2. Input and Output

Input Methods:

1. **Text Field Input:** Users enter task descriptions in the `txtInput` field

2. **Button Clicks:** Users trigger actions by clicking various buttons (Add, Mark Done, Delete, etc.)

3. **List Selection:** Users select tasks from the JList to perform actions on specific tasks

4. **File Dialogs:** For save/load operations, users select files through JFileChooser dialogs

Output Methods:

1. **JList Display:** Tasks are displayed in a scrollable list with completion status and timestamps

2. **Status Label:** Temporary messages appear in `lblStatus` to confirm actions or show errors

3. **Dialog Boxes:** JOptionPane dialogs are used for confirmation (delete, exit) and error messages

4. **File Output:** Tasks are serialized to .todos files when saved

3. Program Components and Implementation

a. ToDo Class (Model)

This class represents individual tasks with:

- **description**: The task text

- **completed**: Boolean flag for completion status

- **creationDate**: When the task was created

- **completionDate**: When the task was marked done (null if not completed)

Key methods:

- **markAsCompleted()/markAsIncomplete()**: Toggle completion status

- **toString()**: Formats task for display with checkbox symbol and timestamps

- Implements **Serializable** for file storage

b. ToDoProManager Class (Controller)

Manages all task operations and maintains the task list. Key features:

1. **Task Management:**

  - ***addTask()***: Validates and adds new tasks

  - ***markTaskAsCompleted()***: Marks selected task as done

  - ***deleteTask()***: Removes selected task

  - ***clearCompletedTasks()***: Removes all completed tasks

2. File Operations:

  - ***saveTasksToFile()***: Serializes tasks to .todos file

  - ***loadTasksFromFile()***: Deserializes tasks from file

  - Uses ***JFileChooser*** for file selection

3. GUI:

  - Maintains a `DefaultListModel` that automatically updates the JList

  - Provides task count information to the GUI

c. NewJFrame Class (View)

The main GUI window built with Java Swing components:

UI Components:

- ***jList1***: Displays all tasks with their status

- ***txtInput***: Text field for entering new tasks

- Buttons for all actions (Add, Mark Done, Delete, etc.)

- ***lblStatus***: Shows operation feedback

Key Features:

- Event handlers for all button actions

- Input validation before operations

- Confirmation dialogs for destructive actions

- Status messages that automatically clear after 3 seconds

- Clean initialization with empty task list

Cool Methods:

- ***addTask()***: Handles task creation workflow

- ***showStatus()***: Displays temporary status messages

- Event handlers for all button actions

4. Technical Implementation Details

File Handling

The program uses Java's serialization mechanism (***ObjectOutputStream/ObjectInputStream***) **to**:

- Save the entire ***ArrayList<ToDo>*** to disk

- Load it back while preserving all task data and statuses

- Automatically handles file extension (.todos)

GUI Implementation

- Built with Java Swing (JFrame, JList, JButtons, etc.)
- Uses DefaultListModel to connect the task list to the JList
- Implements responsive design with proper component layout
- Follows Swing's event-driven programming model

Data Flow

1. User performs action (e.g., clicks "Add" button)
2. Event handler in NewJFrame validates input
3. Calls appropriate ToDoProManager method
4. Manager updates internal data and list model
5. GUI automatically refreshes to show changes
6. Status message displayed to user

5. Advanced Features

1. Status Message Timeout:

- Uses Swing Timer to automatically clear status messages after 3 seconds
- Prevents UI clutter while ensuring feedback is visible

2. Exit Confirmation:

- Checks for unsaved changes when exiting
- Offers save option before quitting

- Handles all three dialog responses (Yes/No/Cancel)

3. Task Display Formatting:

- Shows tasks with [✓] or [ ] for completion status
- Includes creation and completion timestamps
- Uses SimpleDateFormat for consistent date formatting

4. Error Handling:

- Input validation for empty tasks
- Selection checks before operations
- File operation error handling with user feedback

6. Design Patterns and Best Practices

1. Separation of Concerns:

- Clear separation between GUI (View), Manager (Controller), and Task (Model)
- Follows MVC-like architecture

2. Encapsulation:

- All task data is private with proper getters/setters
- Manager handles all task operations

3. Code Organization:

- Proper package structure
- Well-commented code

- Consistent naming conventions

4. User Experience:

- Clear visual feedback for all actions
- Confirmation for destructive operations
- Intuitive interface layout

7. Demonstration of Required Components

The program demonstrates all required components from the rubric:

1. Control Structures:

- Selection: if/else in event handlers
- Iteration: for loops in clearCompletedTasks()

2. File Processing:

- Object serialization for task persistence
- JFileChooser for file selection

3. User-Defined Methods:

- Numerous methods in all classes
- Clear separation of functionality

4. Arrays/Collections:

- ArrayList used to manage tasks

- DefaultListModel for JList integration

8. Potential Enhancements

While the program meets all basic requirements, possible improvements could include:

1. Task prioritization
2. Due dates and reminders
3. Categories or tags for tasks
4. Search/filter functionality
5. Undo/redo capability
6. Alternative storage formats (JSON, XML)