Report: Design and Implementation of a Weekly Planner Application

1. Design Overview

The weekly planner application is designed to manage tasks by assigning them to specific time slots within a workweek. The application handles two types of events: online video calls and in-person meetings. The architecture is modular and expandable, ensuring future types of schedulable items can be added with minimal changes.

2. Class StructureMain Class: Entry point of the application.

- **SchedulerConsole Class**: Command-line interface for user interaction.
- **Workweek Class**: Represents the weekly schedule as a 2x5 array of `WorkEvent` objects.
- **WorkEvent Class**: Abstract class for different types of events.
    - **VideoCall Class**: Represents an online video call with a title and email address.
    - **Meeting Class**: Represents an in-person meeting with a title and coordinates (latitude and longitude).
- **FileUtil Class**: Handles file I/O operations.
- **MalformedStringParameterException Class**: Custom exception for handling malformed string inputs.
- **Enum Types**: `Days` and `Times` enums to represent days of the week and time slots respectively.

3. Design Choices and Justifications

- **Immutable WorkEvent Objects**: Ensures data consistency and prevents accidental modification. Each `WorkEvent` is constructed with all necessary parameters and does not have setter methods. This design choice promotes thread safety and simplifies debugging.
- **Enum Types for Days and Times**: Enums for `Days` and `Times` ensure type safety and limit the possible values, reducing the risk of invalid inputs.
- **Package Structure**: Organizing classes into packages (`core`, `cli`, `ui`, `utils`, `exceptions`) promotes modularity and maintainability. Each package has a clear responsibility, making the codebase easier to navigate and extend.
- **File I/O with Default Path**: The `FileUtil` class includes methods for saving and loading schedules with a predefined default path. This design simplifies user interaction and provides a fallback option.

4. Implementation Details

- **Workweek Class**:
    - Stores the schedule as a 2x5 array of `WorkEvent` objects.
    - Methods include `addToSchedule`, `removeFromSchedule`, `isEmpty`, `getTitleAt`, `getFullDetailsAt`, and `generateWorkweekFromStrings`.

- **SchedulerConsole Class**:
  - Provides an interactive command-line interface for the user.
  - Methods include adding, removing, printing details, loading, and saving events.
- **FileUtil Class**:
  - Methods for reading from and writing to files without using advanced data structures.
- **MalformedStringParameterException Class**:
  - Handles exceptions for malformed strings, ensuring robustness in data handling.

5. Observations and Problematic Design Choices

- **Type Erasure in Strings**: Converting objects to and from strings involves type erasure, which can lead to errors if the string format is not strictly followed. Including a TYPE token in the string format helps identify the specific `WorkEvent` type during reconstruction.
- **Error Handling**: The `MalformedStringParameterException` ensures that malformed inputs are caught early, preventing corrupted data from propagating through the system.
- **Future Expansions**: The current design allows for easy addition of new `WorkEvent` types by extending the abstract class and implementing the required methods.

6. Conclusion

The design and implementation of the weekly planner application achieve the goals of modularity, expandability, and robustness. By adhering to principles of immutability, type safety, and clear package structure, the application is well-prepared for future enhancements and maintenance.