Project Design and Method Specifications for Calendar and Schedule Management Applications

Based on the project requirements, in accordance with the principles of object-oriented programming (OOP) and modular development, this project needs to design the following core classes. The methods of each class only define functional rules (without involving specific implementation logic), ensuring that each class has a single responsibility and a clear structure.

I. Data Model Class

1. Event Class

Core responsibility: To encapsulate the basic data of ordinary events, serving as the carrier for data transmission and storage, containing only attributes and basic access methods, without any business logic.

The required methods:

1. Constructor method

- Parameterless constructor method: public, Create an empty event object.
- Parameterized constructor method (title, description, startDateTime, endDateTime): public, Initialize the core attributes of the event.

1. Getter

- public int getEventId(): Obtain the unique ID of the event
- public String getTitle(): Obtain the event title.
- public String getDescription(): Obtain the event description.
- public LocalDateTime getStartDateTime(): Obtain the start time of the event.
- public LocalDateTime getEndDateTime(): Obtain the end time of the event.

1. Setter

- public void setEventId(int eventId): Set the event ID (only called by the ID generator, not manually modifiable by users).
- public void setTitle(String title): Set the event title.
- public void setDescription(String description): Set the event description.
- public void setStartDateTime(LocalDateTime startDateTime): Set the event start time.
- public void setEndDateTime(LocalDateTime endDateTime): Set the event end time.

1. Extended Field Methods (to adapt to the "Additional Event Fields" supplementary requirement)

- public String getLocation(): Obtain the event location.
- public void setLocation(String location): Set the event location.
- public List<String> getAttendees(): Obtain the list of event attendees.
- public void setAttendees(List<String> attendees): Set the list of event attendees.
- public String getCategory(): Obtain the event category (such as study, work, life).
- public void setCategory(String category): Set the event category.

2. RecurrentEvent Class

Core responsibility: Encapsulate the rule data of recurrent events, associate with ordinary events through eventId, and do not store independent event data.

Required methods:

1. Constructor method

- Parameterless constructor method: public, Create an empty recurrent event rule object.
- Parameterized constructor method (eventId, recurrentInterval, recurrentTimes, recurrentEndDate): public, Initialize the core rules of the recurrent event.

1. Getter

- public int getEventId(): Obtain the ID of the associated ordinary event.
- public String getRecurrentInterval(): Obtain the recurrent interval (such as 1d, 1w, 2w).
- public int getRecurrentTimes(): Obtain the number of recurrences (0 indicates recurrence by end date).
- public LocalDate getRecurrentEndDate(): Obtain the recurrent end date (0 indicates recurrence by times).

1. Setter

- public void setEventId(int eventId): Set the ID of the associated ordinary event.
- public void setRecurrentInterval(String recurrentInterval): Set the recurrent interval.
- public void setRecurrentTimes(int recurrentTimes): Set the number of recurrences.
- public void setRecurrentEndDate(LocalDate recurrentEndDate): Set the recurrent end date.

3. ReminderConfig Class

Core responsibility: Encapsulate the configuration data of event reminders to adapt to the "Reminders/Notifications" supplementary requirement.

Required methods:

1. Constructor method

- Parameterless constructor method: public, Create a default reminder configuration.

- Parameterized constructor method (eventId, remindDuration, isEnable): public, Initialize the reminder rules for the specified event.

1. Getter

- public int getEventId(): Obtain the ID of the associated event.
- public Duration getRemindDuration(): Obtain the advance reminder duration (such as 30 minutes, 1 day).
- public boolean isEnable(): Determine whether the reminder for this event is enabled.

1. Setter

- public void setEventId(int eventId): Set the ID of the associated event.
- public void setRemindDuration(Duration remindDuration): Set the advance reminder duration.
- public void setEnable(boolean enable): Enable/disable the reminder function for this event.

II. Data Access Layer Class

1. FileIOManager Class

Core responsibility: Uniformly manage the read and write operations of local .csv files, implement data persistence, and do not involve business logic processing.

Required methods:

1. Ordinary event file operations

- public void writeEventToCsv(Event event): Write a single event to the event.csv file.
- public List<Event> readAllEventsFromCsv(): Read all ordinary events from event.csv and encapsulate them into an Event list.
- public void updateEventInCsv(Event updatedEvent): Update the specified event in event.csv according to eventId.
- public void deleteEventFromCsv(int eventId): Delete the specified event in event.csv according to eventId.

1. Recurrent event file operations

- public void writeRecurrentEventToCsv(RecurrentEvent recurrentEvent): Write the recurrent event rules to recurrent.csv.
- public List<RecurrentEvent> readAllRecurrentEventsFromCsv(): Read all recurrent event rules from recurrent.csv.
- public void updateRecurrentEventInCsv(RecurrentEvent updatedRecurrentEvent): Update the specified recurrent event rules.
- public void deleteRecurrentEventFromCsv(int eventId): Delete all recurrent rules associated with the specified event.

1. Backup and restore operations (to adapt to the "Backup and Restore" basic requirement)

- public boolean backupAllData(String backupPath): Integrate the data of event.csv and recurrent.csv, back up to a single file at the specified path, and return whether the backup is successful.
- public boolean restoreData(String backupPath, boolean isCoverExisting): Restore data from the specified backup file. If isCoverExisting is true, overwrite the existing data; if false, append data, and return whether the restoration is successful.

1. Reminder configuration file operations (to adapt to supplementary requirements)

- public void writeReminderConfigToFile(ReminderConfig config): Write the reminder configuration to a local file (such as reminder.csv).
- public List<ReminderConfig> readAllReminderConfigs(): Read the reminder configurations of all events.

2. EventIdGenerator Class

Core responsibility: Generate auto-incrementing and unique event IDs to avoid duplicates.

Required methods:

- public int generateNextEventId(): Read the maximum eventId in event.csv and increment by 1 to generate a new ID.

III. Business Logic Layer Class

1. EventManager Class

Core responsibility: Handle core business logic such as event creation, update, deletion, and conflict detection, and call the data access layer to complete persistence.

Required methods:

1. Basic event operations (to adapt to the "Event Creation/Update/Delete" basic requirement)

- public boolean createEvent(Event event, RecurrentEvent recurrentEvent): Create an ordinary event. If it is a recurrent event, save the recurrent rules at the same time, and return whether the creation is successful (parameter validity needs to be verified).
- public boolean updateEvent(Event updatedEvent, RecurrentEvent newRecurrentRule): Update the event and associated recurrent rules, and return whether the update is successful.
- public boolean deleteEvent(int eventId, boolean isDeleteEntireSeries): Delete the event. If isDeleteEntireSeries is true, delete the entire series of recurrent events; if false, delete only the current event, and return whether the deletion is successful.

1. Recurrent event processing (to adapt to the "Recurrent Events" basic requirement)

- public List<Event> generateRecurrentEvents(Event baseEvent, RecurrentEvent recurrentRule): Generate a list of all recurrent event instances according to the base event and

recurrent rules.

1. Conflict detection (to adapt to the "Conflict Detection" supplementary requirement)

- public List<Event> checkEventConflict(Event newEvent): Detect whether there is a time conflict between the new event and existing events, and return the list of conflicting events.

2. SearchManager Class

Core responsibility: Implement basic event search and advanced filtering functions.

Required methods:

1. Basic search (to adapt to the "Event Basic Search" basic requirement)

- public List<Event> searchEventsByDate(LocalDate targetDate): Query all events on the specified date.
- public List<Event> searchEventsByDateRange(LocalDate startDate, LocalDate endDate): Query all events within the specified date range.

1. Advanced filtering (to adapt to the "Event Advanced Search & Filter" supplementary requirement)

- public List<Event> searchEventsByTitle(String keyword): Fuzzy query events according to title keywords.
- public List<Event> filterEventsByCategory(String category): Filter events of the specified category.
- public List<Event> filterEventsByLocation(String location): Filter events at the specified location.

3. ReminderManager Class

Core responsibility: Handle the triggering and configuration management of event reminders to adapt to the "Reminders/Notifications" supplementary requirement.

Required methods:

- public void setReminder(ReminderConfig config): Set reminder rules for the specified event.
- public List<String> getUpcomingReminders(): When the program starts, query upcoming event reminders and return a list of reminder messages (such as "Your next event is coming soon in 30 minutes: Project Meeting").

4. StatisticManager Class

Core responsibility: Implement event data statistics to adapt to the "Event Statistics" supplementary requirement.

Required methods:

- public DayOfWeek getBusiestDayInWeek(): Count the date with the most events in a week.
- public Map<String, Integer> getEventCategoryDistribution(): Count the number of events in each category and return a "category-quantity" mapping.
- public int getMonthlyEventCount(LocalDate month): Count the total number of events in the specified month.
- public double getAverageEventDuration(): Calculate the average duration of all events.

IV. View Presentation Layer Class

1. CalendarView Class

Core responsibility: Implement the calendar view display in CLI mode to adapt to the "Calendar View" basic requirement.

Required methods:

- public void showDailyView(LocalDate targetDate, List<Event> events): Display the daily view of the specified date (such as "Sun 05: Assignment Meeting (11:00)").
- public void showWeeklyView(LocalDate startOfWeek, List<Event> events): Display the weekly view of the specified week.
- public void showMonthlyView(LocalDate targetMonth, List<Event> events): Display the monthly view of the specified month (such as marking events in the date grid).

2. ListView Class

Core responsibility: Implement the event list display in CLI mode.

Required methods:

- public void showEventList(List<Event> events): Display the event list in chronological order, including complete event information.

V. Interface Interaction Class (Optional, to adapt to GUI supplementary requirement)

1. CalendarAppGUI Class

Core responsibility: Serve as the main entrance of theGUI program, integrate various function panels, and manage interface layout and user interaction.

Required methods:

- public void initMainFrame(): Initialize the main window and menu bar.
- public void showEventEditDialog(): Open the event creation/edit pop-up window.
- public void refreshCalendarView(): Refresh the calendar view data.

2. EventEditDialog Class

Core responsibility: Provide the event creation/edit interface in GUI mode.

Required methods:

- public Event getInputEventData(): Obtain the event data entered by the user in the pop-up window and encapsulate it into an Event object.
- public void setEventData(Event event): Echo the data of the event to be edited into the pop-up window components.
- public boolean validateInput(): Verify the validity of the user's input (such as non-empty title, reasonable time logic).

VI. Data Storage Structure

The project uses local .csv files to implement data persistence, splitting storage files by data type to ensure clear data classification and convenient association, adapting to the read and write needs of classes in each layer. The specific storage structure design is as follows:

1. Core Storage Files

- event.csv: Store core data of ordinary events, corresponding one-to-one with the attributes of the Event class, with eventId as the primary key.

Storage fields: eventId(int), title(String), description(String), startDateTime(LocalDateTime), endDateTime(LocalDateTime), location(String), category(String), attendees(String, multiple attendees separated by commas)

- recurrent.csv: Store recurrent event rules, corresponding to the attributes of the RecurrentEvent class, associated with event.csv through eventId.

Storage fields: eventId(int), recurrentInterval(String), recurrentTimes(int), recurrentEndDate(LocalDate)

- reminder.csv: Store event reminder configurations, corresponding to the attributes of the ReminderConfig class, associated with event.csv through eventId.

Storage fields: eventId(int), remindDuration(String, format such as "PT30M", "P1D"), isEnable(boolean)

2. Backup File

- Backup file format: A single file integrating data from event.csv, recurrent.csv, and reminder.csv, stored line by line according to "file identifier-data content" to facilitate distinguishing data types during restoration.

Example format:

EVENT,1,Class,Math Course,2025-10-05T09:00,2025-10-05T10:40,Classroom A,Study,Zhang San,Li Si

RECURRENT,1,1w,4,2025-11-02

REMINDER,1,PT30M,true

3. Data Association Rules

- One-to-one association: event.csv and reminder.csv are associated one-to-one through eventId, and one event corresponds to only one reminder configuration.
- One-to-one association: event.csv and recurrent.csv are associated one-to-one through eventId, and one ordinary event corresponds to only one set of recurrent rules (recurrent event instances are dynamically generated by the business layer and not stored).
- Data consistency guarantee: Add, delete, and modify operations need to synchronize associated files (such as when deleting an event, the corresponding recurrent rules and reminder configurations need to be deleted synchronously), which is uniformly handled by the FileIOManager class.