AbsolutSin-ema Developer Guide

AbsolutSin-ema is a desktop application for party planners to manage contacts and events efficiently. It is optimized for use via a Command Line Interface (CLI) while still having the benefits of a Graphical User Interface (GUI). This guide provides comprehensive documentation for developers who wish to understand, maintain, or extend AbsolutSin-ema.

AbsolutSin-ema is an address book that is designed for party planners across all experience levels — from student organizers and hall committees to freelance coordinators and professional event teams. It serves individuals who manage guests, suppliers, and logistics, and who value fast, organized, and command-driven workflows that streamline event planning and communication.

Table of Contents

Acknowledgements

- This project is based on the AddressBook-Level3 project created by the SE-EDU initiative
- JavaFX library for GUI components
- Jackson library for JSON serialization/deserialization
- JUnit5 for testing framework

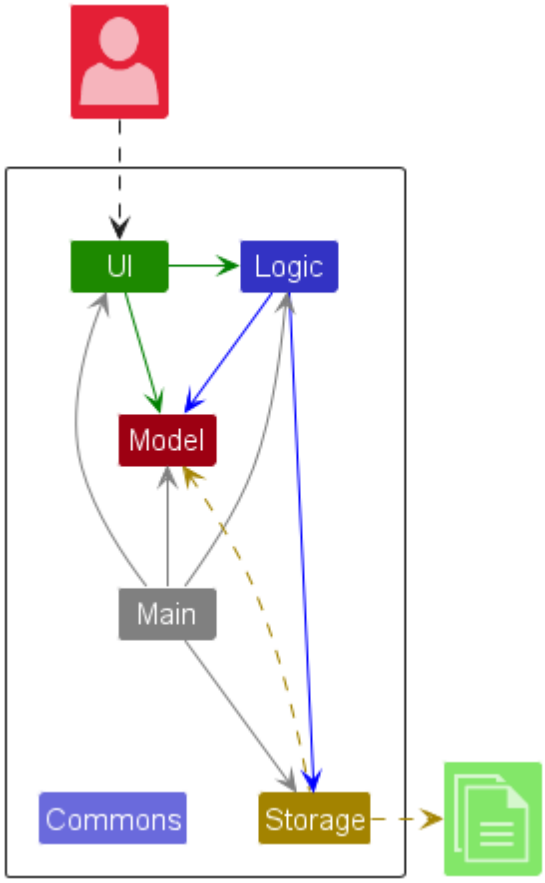
Setting up, getting started

Refer to the guide *Setting up and getting started*.

Design

💡 **Tip:** The `.puml` files used to create diagrams are in this document `docs/diagrams` folder. Refer to the *PlantUML Tutorial* at se-edu/guides to learn how to create and edit diagrams.

Architecture



The ***Architecture Diagram*** given above explains the high-level design of **AbsolutSin-ema**, a specialized contact management application designed for party planners to manage vendors, clients, and events efficiently.

AbsolutSin-ema is built on a **dual-entity system** that manages both **Persons** (vendors/clients) and **Events** (parties) with sophisticated relationship management, budget tracking, and assignment capabilities.

Core Design Principles

1. **Domain-Specific Design**: Built specifically for party planning with vendor management, budget tracking, and event-person assignments
2. **Dual-Entity Architecture**: Separate but interconnected management of Persons and Events
3. **Budget-Aware Operations**: All assignments and operations consider budget constraints
4. **Robust Relationship Management**: PersonId system ensures data integrity across relationships
5. **Command-Driven Workflow**: CLI-first design optimized for power users

Main Components

`Main` (consisting of classes `Main` and `MainApp`) handles app launch and shutdown:

- Initializes components in correct sequence and connects them
- Manages application lifecycle and graceful shutdown
- Handles configuration and logging initialization

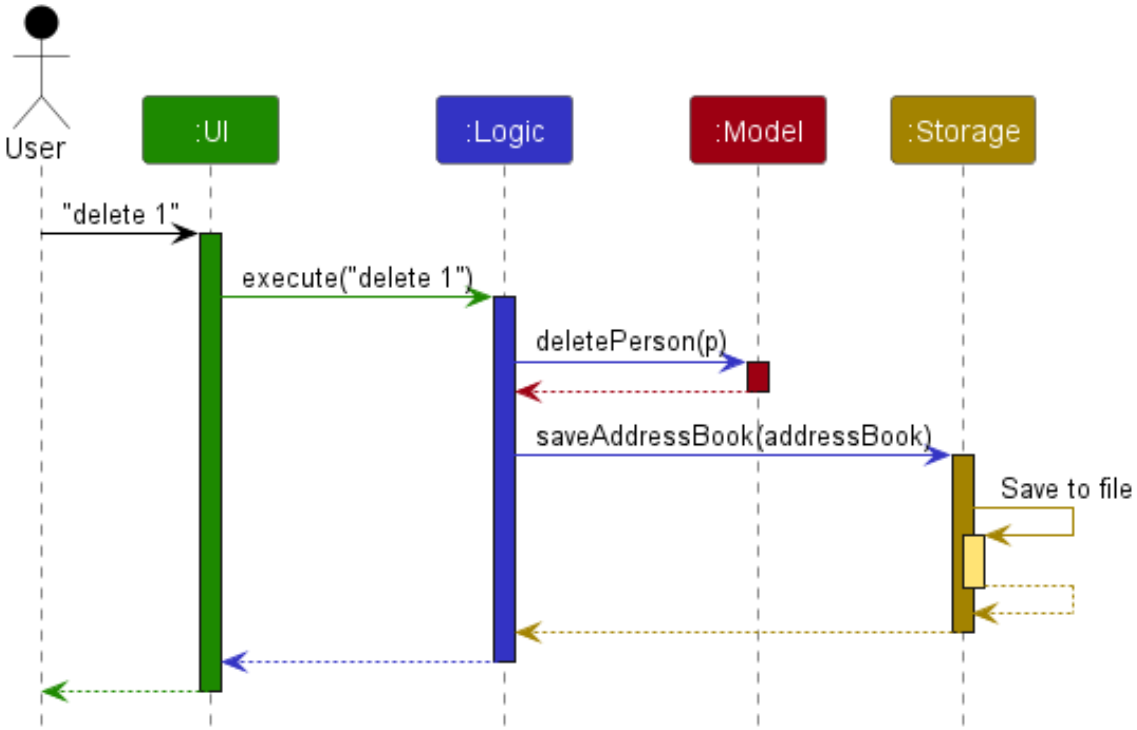
The application's core functionality is delivered through four main components:

- `UI`: The user interface layer
- `Logic`: Command processing and business logic
- `Model`: Data models and business rules
- `Storage`: Data persistence and retrieval

`Commons` provides shared utilities used across components.

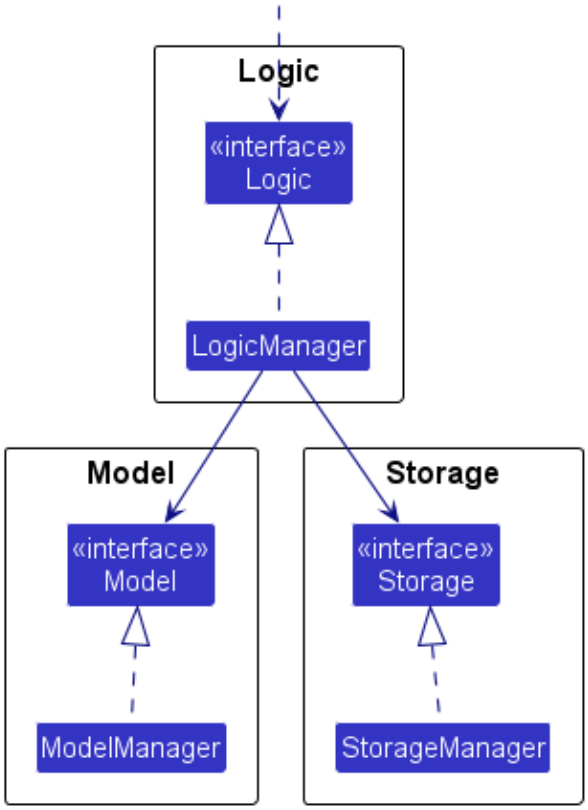
Component Interaction

The *Sequence Diagram* below shows how the components interact for the command `delete 1`:



Each component:

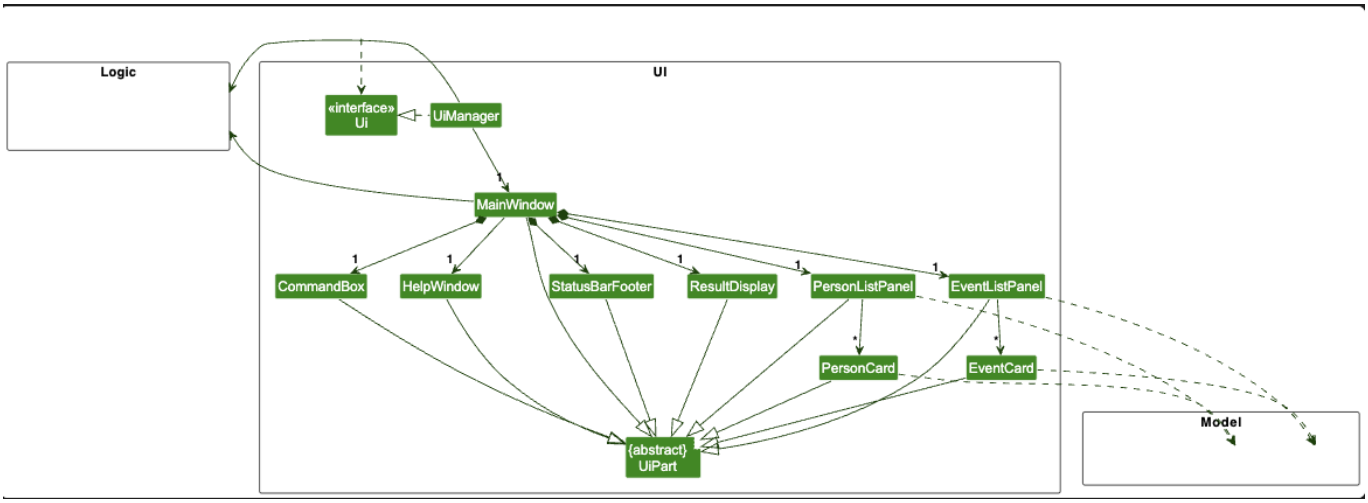
- Defines its API in an `interface` with the same name as the Component
- Implements functionality using a concrete `{Component Name}Manager` class
- Interacts with other components through interfaces to maintain loose coupling



Component Details

UI Component

API: `Ui.java`



The UI consists of a `MainWindow` containing specialized components:

Core UI Components:

- `CommandBox`: Handles CLI input with validation and auto-completion hints
- `ResultDisplay`: Shows command results and error messages
- `PersonListPanel`: Displays vendor/client contacts with tags and budget info
- `EventListPanel`: Shows events/parties with participant counts and budget status

- `StatusBarFooter`: Displays application state and file save status
- `HelpWindow`: Comprehensive help system with scrollable command reference

Key UI Features:

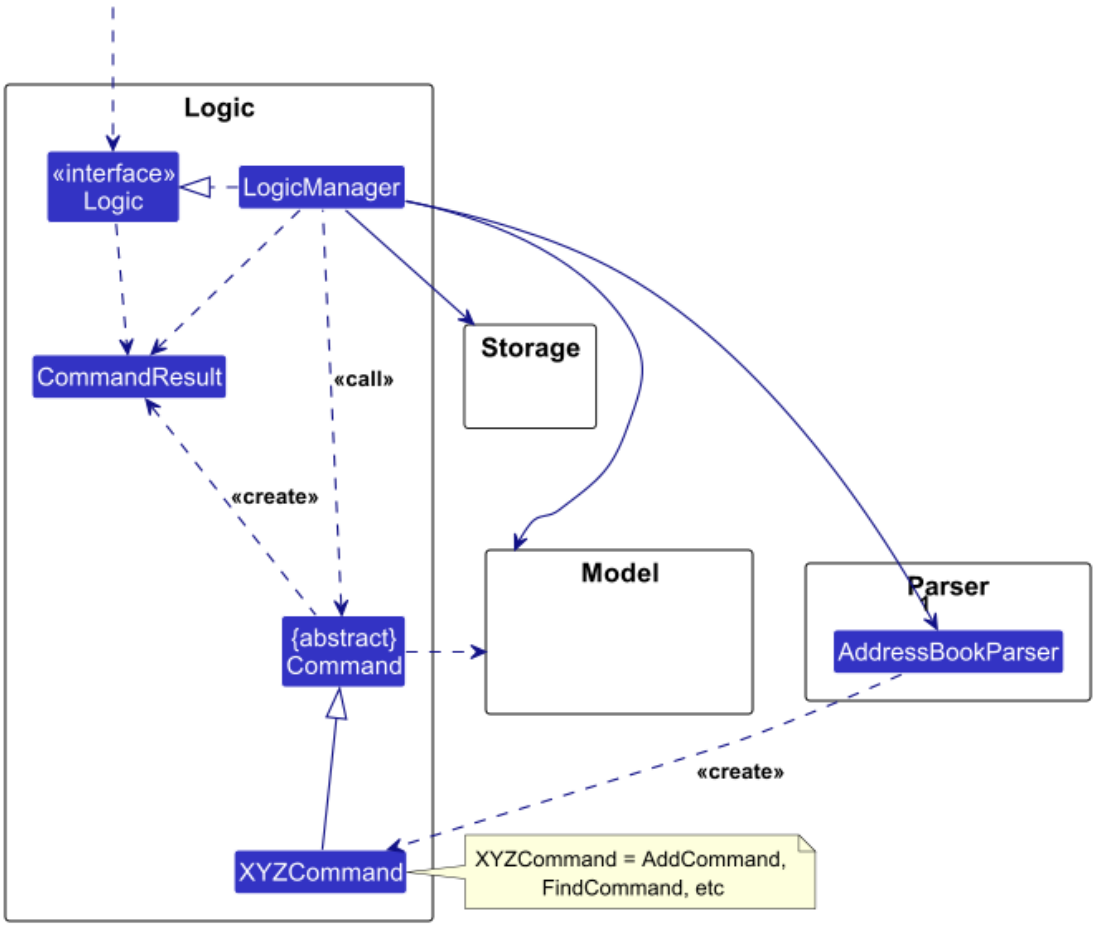
- **Dual-panel design**: Separate views for persons and events
- **Budget visualization**: Color-coded budget indicators in both panels
- **Tag-based filtering**: Visual tag representation for quick identification
- **Real-time updates**: Observable list bindings ensure immediate UI updates
- **Responsive layout**: Adapts to different screen sizes and window states

The UI uses JavaFX framework with FXML layouts stored in `src/main/resources/view`. Each UI component inherits from `UiPart<T>` which provides common functionality for GUI elements.

Logic Component

API : `Logic.java`

Here's the class diagram of the `Logic` component:



Command Processing Flow:

1. `LogicManager` receives user input and delegates to `AddressBookParser`
2. `AddressBookParser` creates appropriate `XYZCommandParser` based on command word
3. Parser validates input and creates `XYZCommand` object
4. `LogicManager` executes command with `Model` interaction

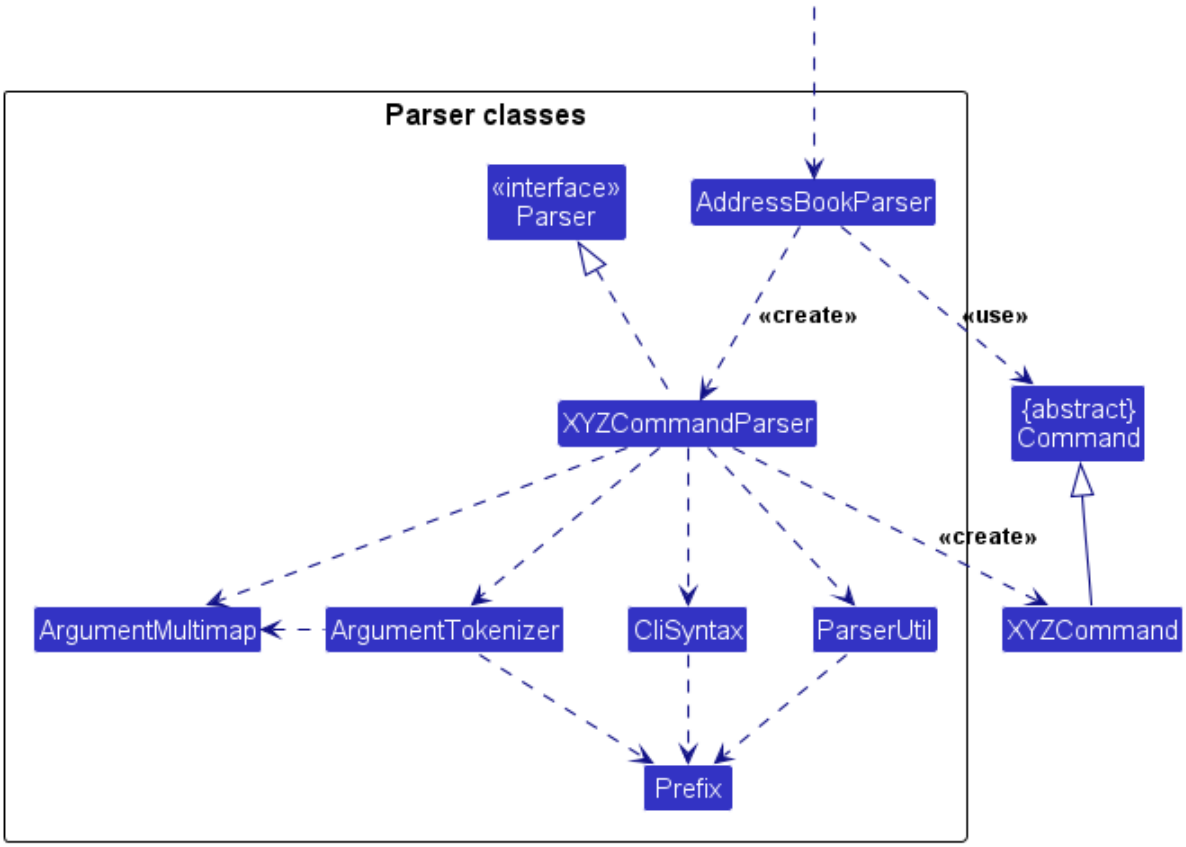
5. Command returns `CommandResult` with success/error information

Specialized Parsers:

- `AddEventCommandParser`: Handles complex event creation with optional contact assignments
- `AssignContactToEventCommandParser`: Manages bulk contact-to-event assignments with budget validation
- `FindCommandParser`: Supports both name and tag-based searching
- `EditCommandParser` / `EditEventCommandParser`: Handle partial updates with validation

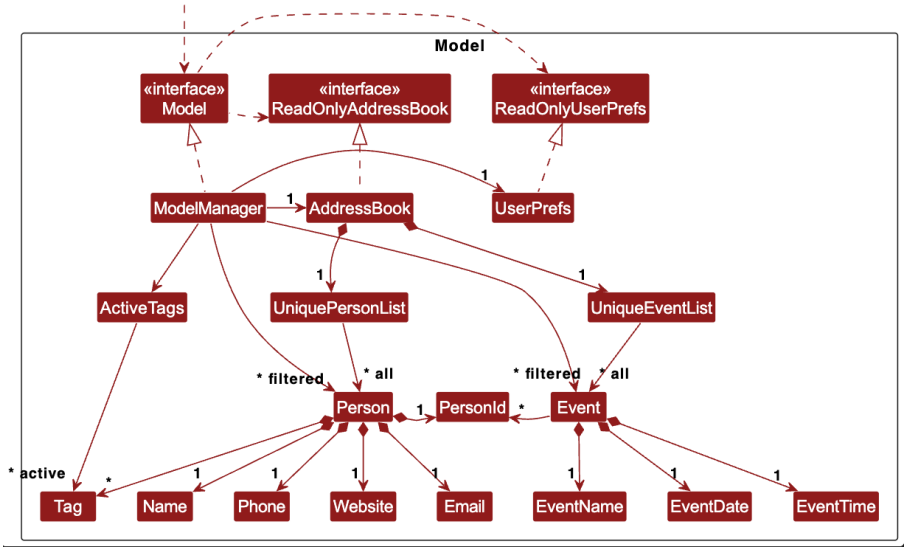
Advanced Features:

- **Budget validation**: Commands validate budget constraints before execution
- **Undo system**: Commands save state for undo functionality via `model.saveStateForUndo()`
- **Index validation**: Robust checking of user-provided indexes against current filtered lists
- **Confirmation flows**: Multi-step commands like `clear` require confirmation



Model Component

API: `Model.java`



Core Model Classes:

`AddressBook`: Central data container managing both persons and events

- `UniquePersonList`: Maintains person data with duplicate prevention
- `UniqueEventList`: Manages events with name-based uniqueness checking
- Deep copying mechanisms for undo functionality

`Person`: Represents vendors/clients with specialized fields

- `PersonId`: Unique identifier for relationship management
- `Name`, `Phone`, `Email`: Standard contact information
- `Website`: Vendor-specific field for online presence
- `Budget`: Vendor service cost for budget calculations
- `Set<Tag>`: Flexible categorization system

`Event`: Represents parties/gatherings with comprehensive management

- `EventName`, `EventDate`, `EventTime`: Basic event information
- `List<PersonId>`: Participants assigned to the event
- `Budget initialBudget`: Total budget allocated for the event
- `Budget remainingBudget`: Budget remaining after assignments

Advanced Model Features:

1. **Budget Management System:**

```
// Budget validation during assignment
if (eventBudget < personBudget) {
    throw new CommandException("Budget exceeded");
}
remainingBudget = eventBudget - personBudget;
```

2. **Relationship Integrity:**

```
// PersonId ensures referential integrity
List<PersonId> participants = event.getParticipants();
// Relationships persist across person edits
```

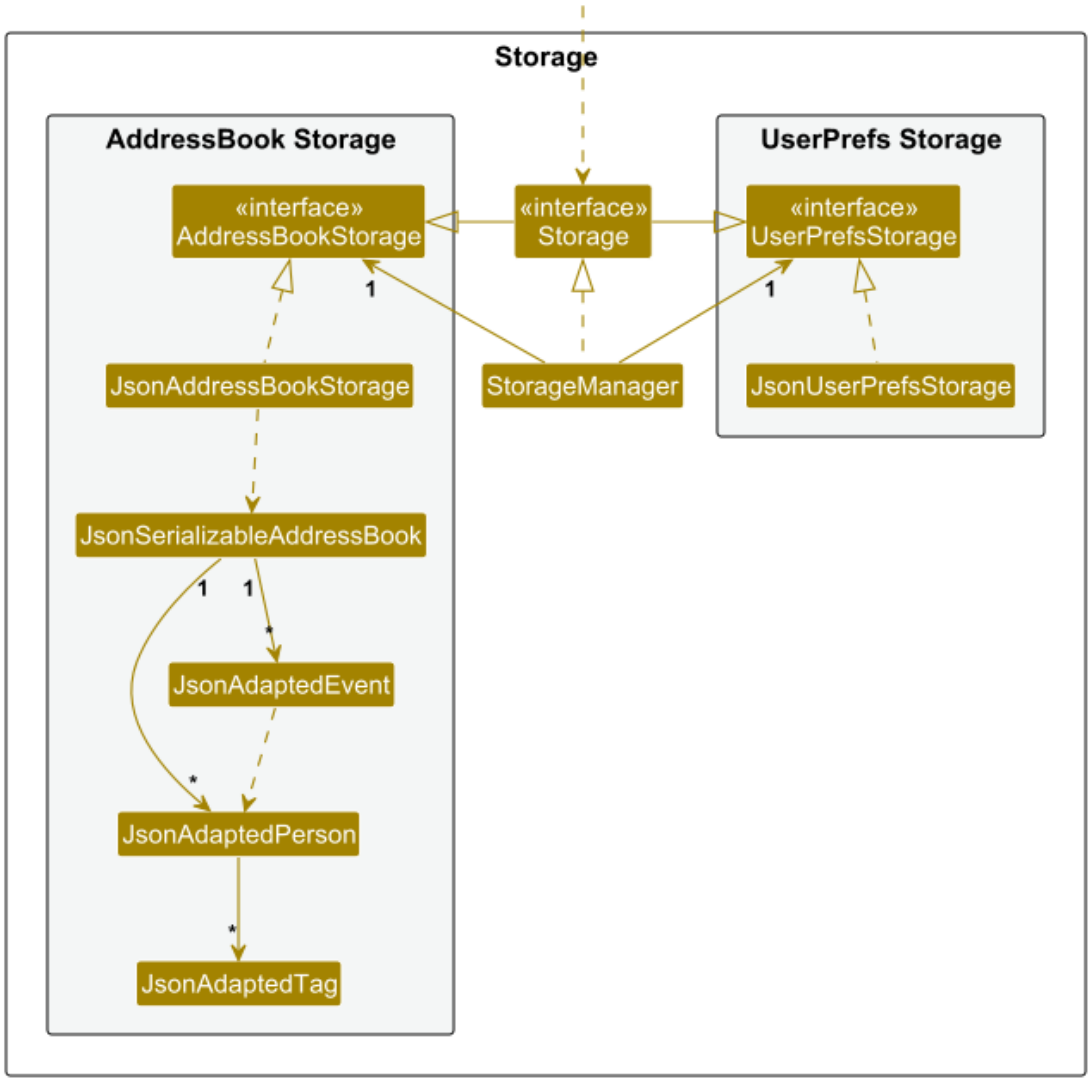
3. **Observable Lists:**

```
ObservableList<Person> getFilteredPersonList();
ObservableList<Event> getFilteredEventList();
// UI automatically updates when data changes
```

The Model is completely independent of UI and Storage components, following clean architecture principles.

Storage Component

API: `Storage.java`



Storage Architecture:

- `StorageManager`: Coordinates between AddressBook and UserPrefs storage
- `JsonAddressBookStorage`: Persists dual-entity data in JSON format
- `JsonUserPrefsStorage`: Manages user preferences and settings

JSON Adapters:

- `JsonAdaptedPerson`: Serializes Person objects with PersonId preservation
- `JsonAdaptedEvent`: Handles Event serialization with participant relationships
- `JsonAdaptedTag`: Manages tag serialization
- `JsonSerializableAddressBook`: Root container for complete data export/import

Key Storage Features:

1. **Automatic Saving**: Data persists after every state-changing command
2. **Graceful Recovery**: Handles corrupted files by clearing data
3. **Relationship Preservation**: PersonId references maintained across sessions
4. **User Preferences**: Window size, position, and view preferences saved

Common classes

Classes used by multiple components are in the `seedu.address.commons` package. These include:

- `LogsCenter`: Handles application-wide logging functionality
- `Config`: Manages application configuration settings
- `Messages`: Contains common message templates and constants
- `JsonUtil`: Provides JSON serialization and file I/O utilities
- `StringUtil`: Common string manipulation and validation helpers
- `AppUtil`: General application utilities and argument validation

---

Implementation

This section describes noteworthy implementation details of key features.

Budget Management System

Overview

AbsolutSin-ema implements sophisticated budget tracking for both individual vendors and events. This system ensures financial constraints are respected when assigning vendors to events.

Budget Components:

- `Budget` class: Validates and stores monetary values
- Event budget tracking: Initial vs. remaining budget separation
- Person budget: Service cost for vendor assignments

Budget Validation Algorithm:

```
public void validateAssignment(Person person, Event event) {
    double personCost = Double.parseDouble(person.getBudget().value);
    double remainingBudget =
Double.parseDouble(event.getRemainingBudget().value);

    if (remainingBudget < personCost) {
        throw new CommandException("Budget exceeded");
    }
}
```

Budget Updates: When assigning contacts to events, the system:

1. Validates each person's cost against remaining budget
2. Updates remaining budget after successful assignment
3. Prevents over-budget assignments
4. Maintains budget history for undo operations

Design Considerations

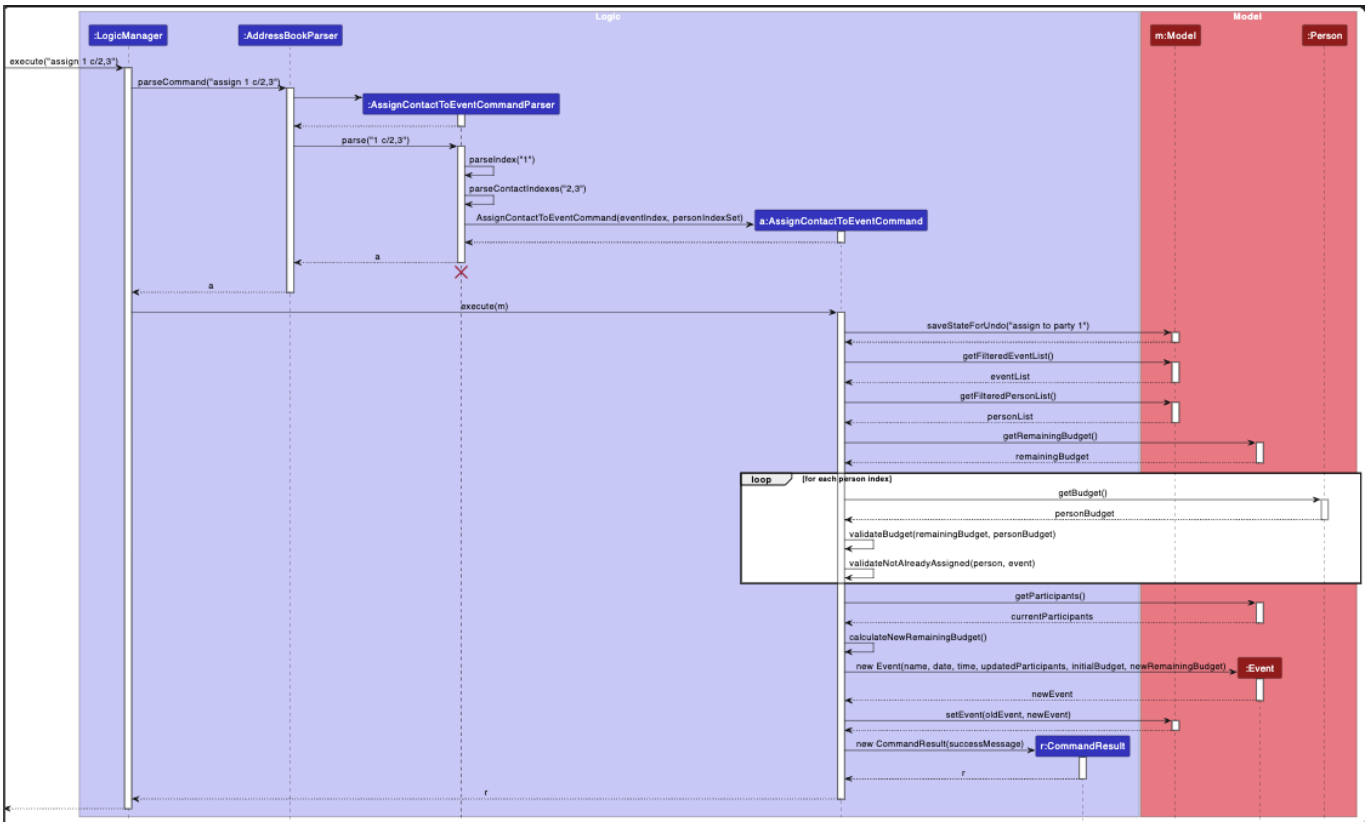
- **Alternative 1 (current choice):** Real-time budget validation
    - Pros: Immediate feedback, prevents budget violations
    - Cons: More complex assignment logic
- **Alternative 2:** Post-assignment budget checking
    - Pros: Simpler implementation
    - Cons: Could allow budget violations

Event-Person Assignment System

Overview

The assignment system manages complex relationships between vendors and events with budget constraints and participant tracking.

The following sequence diagram shows how the assign command works when a user executes `assign 1 c/2,3`:



Assignment Command Flow:

1. `AssignContactToEventCommand` receives event index and person indexes
2. Validates all indexes against current filtered lists
3. Checks for existing assignments to prevent duplicates
4. Validates budget constraints for each assignment
5. Updates event with new participants and adjusted budget
6. Saves state for undo functionality

Key Implementation Details:

```
// Collect and validate persons with budget checking
private List<Person> collectAndValidatePersons(List<Person> lastShownList,
        Event eventToModify, double eventBudget) throws CommandException {
    List<Person> result = new ArrayList<>();
    for (Index i : assignedPersonIndexList) {
        Person personToAdd = lastShownList.get(i.getZeroBased());
        double personBudget =
parseBudgetSafe(personToAdd.getBudget().value);

        if (eventBudget < personBudget) {
            throw new CommandException("Budget exceeded for " +
personToAdd.getName());
        }
        eventBudget -= personBudget;
        result.add(personToAdd);
    }
    return result;
}
```

Assignment Features:

- Bulk assignment: Multiple contacts in single command (`assign 1 c/2,3,4`)
- Budget validation: Automatic checking during assignment
- Duplicate prevention: Cannot assign same person multiple times
- State preservation: Full undo support for assignments

Undo System Implementation

Overview

AbsolutSin-ema implements a state-saving undo system that captures application state before destructive operations.

State Management:

```
// Before executing state-changing command
model.saveStateForUndo("description of operation");

// Execute operation
performOperation();

// Undo restores previous state
model.undo(); // Restores to saved state
```

Undo-Capable Commands:

- Person management: `add`, `delete`, `edit`, `clear`
- Event management: `addp`, `deletep`, `editp`
- Assignments: `assign`, `unassign`

State Saving Strategy:

- Deep copy of entire AddressBook before modification
- Operation description for user feedback
- Single-level undo (most recent operation only)

Design Considerations

- **Current Implementation:**
  - Pros: Simple and reliable
  - Cons: Memory intensive for large datasets
- **Alternative Approach:** Command-specific undo
  - Pros: Memory efficient
  - Cons: Complex implementation, higher error potential

Search and Filter System

Overview

Advanced search capabilities supporting both name-based and tag-based filtering across persons and events.

Search Components:

- `FindCommand`: Unified search across name and tag fields
- `NameAndTagContainsKeywordsPredicate`: Advanced filtering logic
- Real-time result updates in UI

Event Management System

The event management system allows party planners to create and manage events (parties), track budgets, and assign contacts (vendors/clients) to specific events. This feature is fully integrated with the person management system.

Key Components:

1. **Event Entity** (`seedu.address.model.event.Event`)
    - Stores event details: `EventName`, `EventDate`, `EventTime`
    - Manages budgets: `initialBudget` and `remainingBudget`
    - Tracks participants via a list of `PersonId` references
    - Implements `isSameEvent()` method to prevent duplicate events
2. **Event Commands:**
    - `AddEventCommand` (`addp`) - Creates a new event with name, date, time, and budget
    - `EditEventCommand` (`editp`) - Modifies existing event details
    - `DeleteEventCommand` (`deletep`) - Removes an event from the system
    - `AssignContactToEventCommand` (`assign`) - Links one or more contacts to an event as participants (format: `assign EVENT_INDEX c/CONTACT_INDEXES`)
    - `UnassignContactFromEventCommand` (`unassign`) - Removes one or more contacts from an event (format: `unassign EVENT_INDEX c/CONTACT_INDEXES`)
    - `ViewCommand` (`view`) - Displays all participants for a specific event
3. **Budget Tracking:**
    - Each event tracks both initial and remaining budget
    - Each person (vendor) has an associated budget cost
    - When a person is assigned to an event, the remaining budget can be adjusted
    - Budget validation ensures sufficient funds are available
4. **UI Components:**
    - `EventListPanel` - Displays all events in the system
    - `EventListCard` - Shows individual event details including name, date, time, and budget

How Event-Person Association Works:

Events don't store full `Person` objects, but rather `PersonId` references. This design:

- Prevents data duplication
- Ensures person updates automatically reflect in associated events
- Allows efficient querying of participants via `Model#getPersonById(PersonId)`

Example Usage Scenario:

1. Party planner creates a new birthday party: `addp n/Birthday Party d/2024-12-25 t/18:00 b/5000`
2. System creates event with $5000 budget
3. Planner assigns caterer to event: `assign 1 c/1` (assigns contact at index 1 to event at index 1)
4. Planner can assign multiple contacts at once: `assign 1 c/2,3,4` (assigns contacts 2, 3, and 4 to event 1)
5. Planner can view all vendors/participants: `view 1`
6. If needed, planner can unassign: `unassign 1 c/1` (removes contact 1 from event 1)

Confirmation System for Destructive Operations

To prevent accidental data loss, the application implements a confirmation system for the `clear all` command, which deletes all contacts and parties, the `clear parties` command, which deletes only parties, and the `clear contacts`, which deletes only contacts respectively.

How it works:

1. When user executes a clear command, the `ClearCommand` displays a warning message (one of):
    - "Are you sure you want to clear the party planner? (Type 'y' to confirm, 'n' to cancel)" (clear all)
    - "Are you sure you want to clear all contacts? (Type 'y' to confirm, 'n' to cancel)" (clear contacts)
    - "Are you sure you want to clear all parties? (Type 'y' to confirm, 'n' to cancel)" (clear parties)
2. User must type `y` to confirm or `n` to cancel the operation.
3. If confirmed, the `ConfirmClearCommand` performs the actual data deletion and shows one of:
    - "Address book has been cleared!"
    - "Contacts have been cleared!"
    - "Parties have been cleared!"
4. If cancelled, the operation is aborted and no data is lost.

The commands `clear contacts` and `clear parties` work similarly, but just with different warning messages in step 1 that refer to what they are deleting (contacts for `clear contacts` and parties for `clear parties`).

Design Rationale:

- Destructive operations like `clear all`, `clear contacts`, and `clear parties` can be fatal (even with undo, losing all data is risky)
- Interactive confirmation with explicit yes/no response provides a strong safety net
- The confirmation message clearly explains what will happen and how to proceed
- User-friendly prompts reduce the chance of accidental data loss from typos or misclicks

- Could be extended to other dangerous operations (e.g., bulk delete) in future versions

[Proposed] Data archiving

Search Algorithm:

```
public boolean test(Person person) {
    // Name matching (case-insensitive)
    boolean nameMatch = keywords.stream()
        .anyMatch(keyword -> StringUtil.containsWordIgnoreCase(
            person.getName().fullName, keyword));

    // Tag matching
    boolean tagMatch = person.getTags().stream()
        .anyMatch(tag -> keywords.stream()
            .anyMatch(keyword -> StringUtil.containsWordIgnoreCase(
                tag.tagName, keyword)));

    return nameMatch || tagMatch;
}
```

Search Features:

- Multi-keyword support: `find john caterer` matches name OR tag
- Case-insensitive matching
- Partial word matching
- Tag-specific searching: `find caterer` shows all caterers
- Real-time filtering with immediate UI updates

Documentation, logging, testing, configuration, dev-ops

- Documentation guide
- Testing guide
- Logging guide
- Configuration guide
- DevOps guide

Appendix: Requirements

Product scope

Target user profile:

AbsolutSin-ema targets **professional party planners** who:

- Organize multiple events simultaneously (birthdays, anniversaries, corporate events)
- Manage extensive vendor networks (caterers, decorators, entertainers, venues)
- Track budgets across multiple projects

- Need quick access to vendor information during planning
- Prefer efficient CLI tools over slower GUI applications
- Handle time-sensitive vendor assignments and budget allocations

Value proposition: **AbsolutSin-ema** helps party planners manage their contacts more efficiently than generic contact management apps by:

AbsolutSin-ema provides party planners with:

- **Specialized vendor management**: Custom fields (website, budget) for vendor-specific needs
- **Integrated event planning**: Connect vendors to specific events with budget tracking
- **Budget control**: Real-time budget validation prevents overspending
- **Efficient workflow**: CLI commands optimized for rapid data entry and retrieval
- **Relationship management**: Track which vendors work on which events
- **Professional organization**: Tag-based categorization for vendor specialties

---

User stories

Priorities: High (must have) - ∗ ∗ ∗, Medium (should have) - ∗ ∗, Low (nice to have) - ∗

Priority	As a…	I want to…	So that I can…
∗ ∗ ∗	party planner	add vendor contacts	contact the vendor
∗ ∗ ∗	party planner	add vendor contacts with budget info	track service costs for budget planning
∗ ∗ ∗	party planner	create events with budgets	manage individual party finances
∗ ∗ ∗	party planner	assign vendors to specific events	organize who's working on each party
∗ ∗ ∗	party planner	see budget validation during assignments	avoid overspending on events
∗ ∗ ∗	party planner	view all vendors and events in one app	have centralized party planning management
∗ ∗ ∗	party planner	delete individual vendor contacts	remove outdated or incorrect vendor information
∗ ∗ ∗	party planner	delete individual events	remove cancelled or completed parties from my list
∗ ∗	busy party planner	quickly find vendors by specialty tags	locate appropriate services for specific party themes
∗ ∗	party planner	edit vendor and event information	keep information current as details change
∗ ∗	party planner	see confirmation before deleting data	prevent accidental loss of important vendor information

Priority	As a…	I want to…	So that I can…
* *	party planner	clear all contacts at once	reset my contact list when starting fresh
* *	party planner	clear all parties at once	clean up after event season without affecting contacts
* *	party planner	clear all data (contacts and parties)	completely reset the application when needed
* *	party planner	undo recent changes	recover from mistakes during rapid data entry
* *	new user	access comprehensive help	learn the system quickly without external documentation
*	experienced planner	assign multiple vendors to events at once	speed up event setup for large parties
*	budget-conscious planner	see remaining budget for each event	make informed vendor selection decisions
*	organized planner	export vendor and event data	create reports or backup information
*	collaborative planner	share vendor lists with team members	coordinate with assistants and partners

Use cases

(For all use cases below, the **System** is `AbsolutSin-ema` and the **Actor** is the `Party Planner`)

Use case UC01: Add specialized vendor with budget

MSS

1. Planner enters `add n/Elite Catering p/91234567 e/info@elite.com w/www.elite.com t/caterer t/halal b/150`
2. System validates all required fields, ensuring the name is unique and the budget format is correct.
3. System creates a new vendor entry with a unique PersonId.
4. System displays a success message with the new vendor's details.
5. System updates the vendor list displayed in the UI.

   Use case ends.

Extensions

- 2a. Duplicate vendor name detected.

    - 2a1. System shows error message "This person already exists in the address book".
    - Use case ends.
- 2b. Invalid budget format.
    - 2b1. System shows "Budget should only contain numbers, and it should be at least 0."
    - 2b2. Planner corrects input.
    - Use case resumes at step 1.
- 2c. Invalid contact field (phone, email, website):
    - 2c1. System shows specific error message for the invalid field (e.g., "Phone numbers should only contain numbers, and it should be at least 3 digits long").
    - 2c2. Planner corrects input.
    - Use case resumes at step 1.

Use case UC02: Create event and assign vendors with budget validation

MSS

1. Planner enters `addp n/Sarah's Birthday d/15-12-2024 t/19:00 b/500 c/1,3,5`
2. System validates event details and contact indexes.
3. System retrieves vendor costs for contacts 1, 3, and 5.
4. System validates total vendor costs (150+100+75=325) against event budget (500).
5. System creates event with assigned vendors.
6. System updates event remaining budget to 175 (500-325).
7. System displays success message with event and assignment details.
8. System updates the event list displayed in the UI.

   Use case ends.

Extensions

- 2a. Vendor costs exceed event initial budget:
    - 2a1. System shows an error message (e.g., "The total budget of assigned vendors exceeds the event's initial budget.").
    - Use case ends, no event created.
- 2b. Invalid contact index provided:
    - 2b1. System shows an error message (e.g., "The person index provided is invalid").
    - Use case ends, no event created.
- 2c. Invalid event detail format (date, time, budget):
    - 2c1. System shows specific error message for the invalid field.
    - 2c2. Planner corrects input
    - Use case resumes at step 1.
- 2d. Duplicate event name:
    - 2d1. System shows error message "This event already exists in the address book".
    - Use case ends.

Use case UC03: Assign additional vendors to existing event

MSS

1. Planner views the event list and identifies the target event (e.g., at index 2).
2. Planner enters `assign 2 c/4,7`
3. System retrieves the event at the specified index and its current remaining budget.
4. System retrieves the service costs for vendors 4 and 7, and validates them against the event's remaining budget.
5. System checks that vendors 4 and 7 are not already assigned to this event.
6. System updates the event's participant list and adjusts its remaining budget.
7. System displays a success message with the assigned vendor names.
8. System updates the event list displayed in the UI.

   Use case ends.

Extensions

- 2a. Invalid event index:
  - 2a1. System shows error message "The event index provided is invalid".
  - 2a2. Use case ends.
- 2b. Invalid vendor index:
  - 2b1. System shows error message "The person index provided is invalid".
  - 2b2. Use case ends.
- 4a. Insufficient remaining budget
  - 4a1. System shows "The budget of {vendor} exceeds the remaining budget of the party."
  - Use case ends, no assignments made.
- 5a. Vendor already assigned to event.
  - 5a1. System shows "{vendor} has already been assigned to this party."
  - Use case ends.

Use case UC04: Search vendors by specialty/name and assign to event

MSS

1. Planner enters `find photographer` to locate photographers.
2. System filters vendor list showing only vendors whose names or tags contain "photographer".
3. Planner reviews filtered list and identifies suitable vendors (e.g., at index 2 in the filtered list).
4. Planner enters `assign 1 c/2` to assign photographer at index 2 to event 1.
5. System validates assignment (including budget checks) and updates the event's participant list and remaining budget.
6. System displays a success message with the assigned vendor's name and event details.

7. Planner enters `list` to return to full vendor view

   Use case ends.

Extensions

- 1a. Planner searches by multiple keywords (name or tag): find caterer halal
  - 1a1. System filters the vendor list, showing vendors matching "caterer" OR "halal".
  - Use case continues from step 3.
- 2a. No photographer found
  - 2a1. System shows "0 persons listed!"
  - Use case ends
- 5a. Assignment fails due to budget constraints:
  - 5a1. System shows "The budget of {vendor} exceeds the remaining budget of the party."
  - Use case ends, no assignment is made.
- 5b. Assignment fails due to invalid event or vendor index:
  - 5b1. System shows an appropriate error message (e.g., "The event index provided is invalid" or "The person index provided is invalid").
  - Use case ends, no assignment is made.

Use case UC05: Edit vendor information.

MSS

1. Planner identifies the vendor to edit (e.g., at index 1).
2. Planner enters `edit 1 p/87654321 b/250`
3. System validates the new information (e.g., phone number format, budget value).
4. System updates the vendor's details.
5. System displays a success message confirming the changes.
6. System updates the vendor list displayed in the UI.

   Use case ends

Extensions

- 2a. Invalid vendor index:
  - 2a1. System shows error message "The person index provided is invalid".
  - Use case ends.
- 3a. Invalid format for any field:
  - 3a1. System shows specific error message for the invalid field.
  - Use case ends, no changes are made to the vendor.

Use case UC06: Delete a vendor

MSS

1. Planner identifies the vendor to delete (e.g., at index 1).

2. Planner enters `delete 1`
3. Planner confirms deletion(UC15)
4. System removes the vendor from the address book.
5. System displays a success message confirming the deletion.
6. System updates the vendor list displayed in the UI.

   Use case ends.

Extensions

- 2a. Invalid vendor index:
  - 2a1. System shows error message "The person index provided is invalid".
  - Use case ends.

Use case UC07: List all vendors

MSS

1. Planner enters `list`
2. System displays an unfiltered list of all vendors in the UI.

   Use case ends.

Extensions

- 2a. No vendors exist in the system:
  - 2a1. System displays message "0 persons listed!".
  - Use case ends.

Use case UC08: Edit event information

MSS

1. Planner identifies the event to edit (e.g., at index 1).
2. Planner enters `editp 1 n/Wedding Reception d/26-06-2025 b/3000`
3. System validates the new event details.
4. System updates the event's information, including recalculating remaining budget if the initial budget changed and re-validating against assigned vendor costs.
5. System displays a success message confirming the changes.
6. System updates the event list displayed in the UI.

   Use case ends.

Extensions

- 2a. Invalid event index:
  - 2a1. System shows error message "The event index provided is invalid".
  - Use case ends.
- 3a. Invalid format for any field (date, time, budget):
  - 3a1. System shows specific error message for the invalid field.
  - Use case ends, no changes are made to the event.
- 4a. New initial budget is less than total assigned vendor costs:
  - 4a1. System shows error message "The new budget is less than the total cost of currently assigned vendors."
  - Use case ends, no changes are made.

Use case UC09: Delete an event

MSS

1. Planner identifies the event to delete (e.g., at index 1).
2. Planner enters `deletep 1`
3. Planner confirms deletion(UC16).
4. System removes the event from the address book.
5. System displays a success message confirming the deletion.
6. System updates the event list displayed in the UI.

   Use case ends.

Extensions

- 2a. Invalid event index:
  - 2a1. System shows error message "The event index provided is invalid".
  - Use case ends.

Use case UC10: View event participants

MSS

1. Planner identifies the event (e.g., at index 1).
2. Planner enters `view 1`
3. System displays a detailed view of Event 1, including its name, date, time, budget, and a list of all assigned vendors (PersonId to Name/details).

   Use case ends.

Extensions

- 2a. Invalid event index:
  - 2a1. System shows error message "The event index provided is invalid".
  - Use case ends.

- 3a. No vendors assigned to the event:
  - 3a1. System displays event details and message "No vendors assigned to this event."
  - Use case ends.

Use case UC11: List all events

MSS

1. Planner enters `listp`
2. System displays an unfiltered list of all events in the UI.

   Use case ends.

Extensions

- 2a. No events exist in the system:
  - 2a1. System displays message "0 events listed!".
  - Use case ends.

Use case UC12: Unassign contact from event

MSS

1. Planner identifies the event and the vendors to unassign (e.g., event index 1, contacts at index 2 and 3).
2. Planner enters `unassign 1 c/2,3`
3. System retrieves the event and the specified vendors.
4. System removes vendors 2 and 3 from the event's participant list.
5. System adjusts the event's remaining budget by adding back the unassigned vendors' costs.
6. System displays a success message confirming the unassignment.
7. System updates the event list displayed in the UI.

   Use case ends.

Extensions

- 2a. Invalid event index:
  - 2a1. System shows error message "The event index provided is invalid".
  - Use case ends.
- 2b. Invalid vendor index:
  - 2b1. System shows error message "The person index provided is invalid".
  - Use case ends.
- 4a. Vendor not assigned to the event:
  - 4a1. System shows message "{vendor} is not assigned to this party."
  - Use case ends.

Use case UC13: Undo recent operation

MSS

1. Planner performs a state-changing command (e.g., add, delete, assign).
2. Planner enters `undo`
3. System restores the application state to reflect the state before the last command was executed.
4. System displays a success message (e.g., "Undid {description of undone operation}").
5. System updates the UI to reflect the restored state.

   Use case ends.

Extensions

- 2a. No previous operation to undo:
  - 2a1. System shows error message "There is no command to undo!".
  - Use case ends.

Use case UC14: Clear all data with confirmation

MSS

1. Planner enters `clear all`
2. System displays a warning message: "Are you sure you want to clear the party planner? (Type 'y' to confirm, 'n' to cancel)".
3. Planner enters y to confirm.
4. System deletes all vendors and events.
5. System displays a success message "Address book has been cleared!".
6. System updates the UIs (vendor list and event list) to be empty.

   Use case ends.

Extensions

- 3a. Planner enters n to cancel:
  - 3a1. System displays "Operation cancelled."
  - Use case ends, no data is deleted.
- 3b. Planner enters an invalid input (neither 'y' nor 'n'):
  - 3b1. System re-prompts for confirmation.
  - Use case resumes at step 3.

Use case UC15: Clear only contacts with confirmation

MSS

1. Planner enters `clear contacts`

2. System displays a warning message: "Are you sure you want to clear all contacts? (Type 'y' to confirm, 'n' to cancel)".
3. Planner enters y to confirm.
4. System deletes all contact/vendor entries.
5. System displays a success message "All contacts have been cleared!".
6. System updates the vendor list in the UI to be empty.

   Use case ends.

Extensions

- 3a. Planner enters n to cancel:
  - 3a1. System displays "Operation cancelled."
  - Use case ends, no contacts are deleted.
- 3b. Planner enters an invalid input (neither 'y' nor 'n'):
  - 3b1. System re-prompts for confirmation.
  - Use case resumes at step 3.

Use case UC16: Clear only events with confirmation

MSS

1. Planner enters `clear parties`
2. System displays a warning message: "Are you sure you want to clear all parties? (Type 'y' to confirm, 'n' to cancel)".
3. Planner enters y to confirm.
4. System deletes all event entries.
5. System displays a success message "All parties have been cleared!".
6. System updates the event list in the UI to be empty.

   Use case ends.

Extensions

- 3a. Planner enters n to cancel:
  - 3a1. System displays "Operation cancelled."
  - Use case ends, no events are deleted.
- 3b. Planner enters an invalid input (neither 'y' nor 'n'):
  - 3b1. System re-prompts for confirmation.
  - Use case resumes at step 3.

Use case UC17: Find events by name

MSS

1. Planner enters `findp birthday`
2. System filters the event list to show only events whose names contain "birthday".
3. System displays the filtered list of events in the UI.

   Use case ends.

Extensions

- 2a. No events found matching the keywords:
  - 2a1. System shows "0 events listed!".
  - Use case ends.

Use case UC18: Access comprehensive help

MSS

1. Planner enters `help`
2. System opens a HelpWindow displaying comprehensive instructions, command formats, and examples.
3. The HelpWindow is scrollable and allows the planner to navigate through the help content.

   Use case ends.

Non-Functional Requirements

Performance Requirements

1. Command execution should complete within 2 seconds on a standard desktop
2. Application should handle up to 1000 vendors and 100 events without performance degradation
3. Search results should appear within 1 second for any query
4. UI updates should be immediate when data changes

Usability Requirements

1. New users should successfully create vendors and events within 10 minutes using help
2. Error messages should clearly indicate the problem and suggest fixes
3. CLI commands should follow consistent syntax patterns
4. GUI should be readable on screen resolutions from 1024x768 upward

Reliability Requirements

1. Data should auto-save after every successful command
2. Application should recover gracefully from corrupted data files
3. Budget calculations should be accurate to 2 decimal places
4. Undo functionality should reliably restore previous state

Compatibility Requirements

1. Should run on Windows 10+, macOS 10.15+, and Ubuntu 18.04+

2. Requires Java 17 or higher
3. Should work offline without internet connectivity
4. Data files should be portable across operating systems

Maintainability Requirements

1. New command types should be addable without modifying existing commands
2. New vendor fields should be addable through configuration
3. Code should follow established design patterns for consistency
4. Component interfaces should remain stable across versions

Reliability Constraints

1. Budget values must always be formatted and displayed with exactly 2 decimal places
2. Assigning contacts must not cause the total cost to exceed the party's budget
3. Editing a contact's budget must not violate any party budget constraints for parties they are assigned to
4. A contact cannot be assigned to multiple parties occurring at the same date and time
5. Duplicate contact names are not allowed (case-insensitive comparison)

Usability Constraints

1. Budget values must contain up to 7 digits with up to 2 decimal places
2. Phone numbers must be between 3 to 15 digits long
3. Event dates cannot be in the past (must be present or future dates)
4. Search operations are case-insensitive for names and tags
5. Find command treats spaces as keyword separators (not phrase search)
6. Tag editing does not support cumulative addition - all existing tags must be replaced when editing
7. Only one level of undo is supported (no redo functionality)
8. Undo command cannot undo another undo command

Glossary

Term	Definition
AbsolutSin-ema	The name of this party planning contact and event management application
Party Planner	Professional who organizes and coordinates events and parties
Vendor/Person/Contact	Service provider such as caterer, decorator, entertainer, venue, photographer
Event/Party	A planned gathering such as birthday, anniversary, wedding, corporate event
Assignment	The process of allocating vendors to work on specific events
Budget	Monetary amount allocated for vendor services or event expenses
Tag	Label used to categorize vendors by specialty (e.g., caterer, halal, elegant)
PersonId	Unique identifier ensuring data integrity across vendor-event relationships

Term	Definition
CLI	Command Line Interface - text-based commands for rapid data manipulation
GUI	Graphical User Interface, displays contacts and events visually
Index	Numerical position of vendor/event in the currently displayed list
Service Type	Category of vendor service (caterer, DJ, venue, etc.)
Party Theme	Style of a party (princess, superhero, elegant, tropical)
MVP	Minimum Viable Product, core features needed for release
Remaining Budget	Event budget minus already assigned vendor costs
Bulk Assignment	Assigning multiple vendors to an event in a single command

Appendix: Instructions for manual testing

Given below are instructions to test AbsolutSin-ema manually.

ℹ️ Note: These instructions provide a starting point for testers. Testers are expected to do more exploratory testing beyond these scenarios.

Launch and shutdown

1. **Initial launch**
    1. Download the jar file and copy into an empty folder
    2. Open a command terminal, `cd` into the folder you put the jar file in, and use the `java -jar absolutsinema.jar` command to run the application.
    3. **Expected**: GUI appears with sample vendors and events. Window may not be optimally sized.
2. **Saving window preferences**
    1. Resize window to a comfortable size and move to preferred location
    2. Close the window
    3. Re-launch the app by re-typing `java -jar absolutsinema.jar` in the terminal
    4. **Expected**: Window appears in the same size and location as before closing

Vendor management

1. **Adding a vendor with all fields**
    1. Test case: `add n/John's Catering p/98765432 e/john@catering.com w/www.johnscatering.com t/caterer t/halal b/200`
    2. **Expected**: New vendor appears in list with all specified details. Success message shows vendor details.

3. Test case: `add n/John's Catering p/91234567 e/different@email.com w/different.com t/caterer b/150`
4. **Expected**: Error message: "This person already exists in the address book".
5. Test case: `add n/Invalid p/13 e/bad-email w/bad-url t/test b/-50`
6. **Expected**: Error messages for invalid phone, email, website, or negative budget.

2. **Editing vendor information**
    1. Prerequisites: At least one vendor in the list
    2. Test case: `edit 1 p/87654321 b/250`
    3. **Expected**: First vendor's phone and budget updated. Success message displays changes.
    4. Test case: `edit 0 n/Invalid Name`
    5. **Expected**: Error message indicating invalid command format (index out of bounds).
3. **Deleting a vendor (with confirmation)**
    1. Prerequisites: At least one vendor
    2. Test case: `delete 1`
    3. **Expected**: Warning dialog with message: "Are you sure you want to delete this person? (Type 'y' to confirm, 'n' to cancel)"
    4. Confirm in the dialog.
    5. **Expected**: Vendor 1 is removed. Success message: "Deleted Person: ..."
4. **Listing vendors**
    1. Test case: `list`
    2. **Expected**: Person list resets to all persons. Message: "Listed all persons"
5. **Listing all tags**
    1. Test case: `listtags`
    2. **Expected**: If tags exist, message starts with "Listed all tags:" followed by a sorted list. If none, "No tags found in the address book."

Event management

1. **Creating events with budget**
    1. Test case: `addp n/Birthday Party d/25-12-2026 t/18:00 b/1000`
    2. **Expected**: New event created with specified budget. Event appears in event list. Success: "New party added: ..."
    3. Test case: `addp n/Birthday Party d/25-12-2026 t/18:00 b/500`
    4. **Expected**: Error message: "This party already exists in the party list"

5. Test case: `addp n/Wedding d/32-01-2026 t/25:00 b/abc`
6. **Expected**:
    - Date: "Dates should be in the format dd-MM-yyyy, must be a valid calendar date, and cannot be before today."
7. Test case: `addp n/Wedding d/30-01-2026 t/25:00 b/abc`
8. **Expected**:
    - Time: "Times should be in the format HH:mm"
9. Test case: `addp n/Wedding d/30-01-2026 t/21:00 b/abc`
10. **Expected**:
    - Budget: "Budget should only contain up to 7 digits with up to 2 decimal places, and it should be at least 0."
11. Test case: create in the past (use a past date/time): `addp n/Past Party d/01-01-2020 t/00:10 b/100`
12. **Expected**: Error message: "Events cannot be scheduled in the past."

2. **Creating events with initial vendor assignments**
    1. Prerequisites: At least 3 vendors with known budgets
    2. Test case: `addp n/Corporate Event d/15-01-2027 t/14:00 b/2000 c/1,2,3`
    3. **Expected**: Event created; vendors 1, 2, 3 assigned. Remaining budget calculated correctly.
    4. Test case: `addp n/Small Party d/20-01-2027 t/12:00 b/100 c/1,2,3`
    5. **Expected**: Error message for exceeding budget of any contact during creation: "The budget of {Name} exceeds the remaining budget of the party."
    6. Test case (concurrent assignment on same date):
        - Create Event A on date D.
        - Assign vendor X to Event A.
        - Create Event B on the same date D (time can differ).
        - Add vendor X in creation: `addp n/Event B d/<D> t/18:00 b/500 c/<index of X>`
        - **Expected**: Error: "{Name} is already assigned to another party on the same date."

3. **Editing event information**
    1. Prerequisites: At least one event
    2. Test case: `editp 1 n/Updated Name t/19:30`
    3. **Expected**: Success: "Edited Party: ..."
    4. Test case (no fields): `editp 1`

5. **Expected**: Error message: "At least one field to edit must be provided."
6. Test case (move past event without changing date/time):
    - If event 1 is already in the past, run: `editp 1 n/Name Only`
    - **Expected**: "Events cannot be scheduled in the past. Please choose a date and time that is now or in the future."
7. Test case (reduce budget below assigned sum):
    - Ensure event 1 has participants with total cost S.
    - Run: `editp 1 b/<S-1>`
    - **Expected**: "The new budget is less than the total budget of assigned contacts."

4. **Deleting an event (with confirmation)**
    1. Prerequisites: At least one event
    2. Test case: `deletep 1`
    3. **Expected**: Warning dialog with message: "Are you sure you want to delete this party?"
    4. Confirm in the dialog.
    5. **Expected**: Event 1 removed. Success message: "Deleted Party: ..."

Assignment system

1. **Assigning vendors to events**
    1. Prerequisites: Events and vendors exist; note remaining budget
    2. Test case: `assign 1 c/2,3`
    3. **Expected**: Vendors 2 and 3 assigned to event 1. Budget updated. Success message: "Assigned the following people to {Event Name}'s party: {Names}"
    4. Test case: `assign 1 c/2`
    5. **Expected**: If vendor 2 already assigned to event 1: "{Name} has already been assigned to this party."
    6. Test case: `assign 1 c/99`
    7. **Expected**: Error message: "The person index provided is invalid Try switching to the person list view and retry."
    8. Test case (duplicate prefix): `assign 1 c/2 c/3`
    9. **Expected**: Error message: "Invalid command format! \nPlease only include one prefix c/!"
2. **Budget validation during assignment**
    1. Prerequisites: Event with low remaining budget, and a vendor whose cost exceeds it
    2. Test case: `assign 1 c/5` (where vendor 5's cost > event 1 remaining budget)
    3. **Expected**: Error: "The budget of {Name} exceeds the remaining budget of the party."

3. **Prevent scheduling conflicts on the same date**

    1. Prerequisites: Two events on the same date, vendor X not yet on event 2
    2. Assign vendor X to event 1, then run `assign 2 c/<index of X>`
    3. **Expected**: Error: "{Name} is already assigned to another party on the same date."

4. **Unassigning vendors from events**

    1. Prerequisites: Event 1 has assigned vendors
    2. Test case: `unassign 1 c/2`
    3. **Expected**: Vendor 2 removed from event 1. Remaining budget increases appropriately. Success message: "Unassigned the following people from {Event Name}'s party: {Names}"
    4. Test case (not assigned): `unassign 1 c/99` (99 refers to a person that exists but is not assigned)
    5. **Expected**: "{Name} is not assigned to this party."
    6. Test case (invalid person index in current view): `unassign 1 c/999`
    7. **Expected**: "The person index provided is invalid Try switching to the person list view and retry."

Viewing participants of an event

1. **Viewing**

    1. Prerequisites: Events exist
    2. Test case: `view 1`
    3. **Expected**: Error message: "The event index provided is invalid."
    4. Test case: `view 0`
    5. **Expected**: Error message: "Invalid index for view command. Usage: view INDEX (must be a positive integer)"

Search and filter

1. **Finding vendors by name and tags**

    1. Test case: `find caterer`
    2. **Expected**: Shows all vendors tagged with "caterer".
    3. Test case: `find john`
    4. **Expected**: Shows vendors with "john" in their name.
    5. Test case: `find caterer photographer`

6. **Expected**: Shows vendors tagged with either "caterer" OR "photographer". Message shows count, e.g., "2 persons listed!"
7. Test case: `find nonexistent`
8. **Expected**: Shows "0 persons listed!"

Undo functionality

1. **Undoing recent operations**
    1. Prerequisites: Perform any state-changing command (add, edit, assign, etc.)
    2. Test case: `undo`
    3. **Expected**: Previous operation reversed. Message starts with "Previous command undone: ..."
    4. Test case: `undo` (when no previous operation exists)
    5. **Expected**: Error message: "No command to undo."

Clear operations (with confirmation)

1. **Clear contacts only**
    1. Test case: `clear contacts`
    2. **Expected**: Warning dialog: "Are you sure you want to clear all contacts? (Type 'y' to confirm, 'n' to cancel)"
    3. Confirm in the dialog.
    4. **Expected**: Message: "Contacts have been cleared!" and the contacts list is empty (events remain).
2. **Clear parties only**
    1. Test case: `clear parties`
    2. **Expected**: Warning dialog: "Are you sure you want to clear all parties? (Type 'y' to confirm, 'n' to cancel)"
    3. Confirm in the dialog.
    4. **Expected**: Message: "Parties have been cleared!" and the events list is empty (contacts remain).
3. **Clear all data**
    1. Test case: `clear all`
    2. **Expected**: Warning dialog: "Are you sure you want to clear the party planner? (Type 'y' to confirm, 'n' to cancel)"
    3. Confirm in the dialog.
    4. **Expected**: Message: "Address book has been cleared!" and both lists are empty.

Data persistence

1. **Handling corrupted data files**

1. Exit the application
2. Edit the JSON data file and introduce syntax errors
3. Restart the application
4. **Expected**: Invalid data file popup appears, application starts with cleared data

2. **Data saving verification**

1. Make several valid changes (add vendors, create parties, assignments)
2. Exit and restart application
3. **Expected**: All changes preserved exactly as made.

3. **Invalid command formats**

1. Test case: `add` (missing parameters)
2. **Expected**: Error: "Invalid command format!" followed by usage for `add`.
3. Test case: `assign` (missing parameters)
4. **Expected**: Error: "Invalid command format!" followed by usage for `assign`.

- For contact indexes inside `c/` lists, a zero value triggers: "Index is not a non-zero unsigned integer.":

5. Test case: `invalidcommand`
6. **Expected**: Error message: "Unknown command"

4. **Index boundary testing**

1. Prerequisites: Note the current list sizes for vendors (N persons) and events (M events)
2. Test case: `delete 0`

- **Expected**:

```
Invalid command format!
delete: Deletes the person identified by the index number
used in the displayed person list.
Parameters: INDEX (must be a positive integer)
Example: delete 1
```

3. Test case: `delete X` where X > N (exceeds person list size)

- **Expected**: "The person index provided is invalid"

4. Test case: `deletep 0`

- **Expected**:

```
Invalid command format!
deletep: Deletes the party identified by the index number
used in the displayed party list.
```

```
Parameters: INDEX (must be a positive integer)
Example: deletep 1
```

5. Test case: `deletep Y` where Y > M (exceeds event list size)
    - **Expected**: "The event index provided is invalid"
6. Test case: `assign 0 c/1`
    - **Expected**:

```
Invalid command format!
assign: Assigns one or more contacts to a specific party.
Parameters: PARTY_INDEX (must be a positive integer)
c/CONTACT_INDEX[,CONTACT_INDEX...]
Example: assign 1 c/2,3
```

7. Test case: `assign 1 c/0`
    - **Expected**: "Index is not a non-zero unsigned integer."
8. Test case: `assign Y c/1` where Y > M (exceeds event list size)
    - **Expected**: "The event index provided is invalid"
9. Test case: `assign 1 c/X` where X > N (exceeds person list size)
    - **Expected**: "The person index provided is invalid Try switching to the person list view and retry."

Help and exit

1. **Help window**
    1. Test case: `help`
    2. **Expected**: Help window opens. Message: "Opened help window."
2. **Exit**
    1. Test case: `exit`
    2. **Expected**: Application closes.

These test cases cover the major functionality of AbsolutSin-ema. Testers should also explore edge cases, rapid command sequences, and integration scenarios to ensure robust operation.

Appendix: Effort

Overview

Developing AbsolutSin-ema required significant effort beyond the base AddressBook-Level3 project. The application evolved from a simple contact manager to a sophisticated dual-entity system designed specifically for party planning professionals.

Major Enhancements

1. Dual-Entity Architecture (High Effort)

Challenge: Extending from single-entity (Person) to dual-entity (Person + Event) system **Implementation Effort**:

- Created complete Event model with EventName, EventDate, EventTime classes
- Developed UniqueEventList with duplicate detection and management
- Modified AddressBook to handle both entities simultaneously
- Updated Storage layer to persist both entities in JSON format
- Enhanced UI to display both persons and events effectively

Lines of Code: ~800 additional lines across model, storage, and UI layers

2. Event-Person Relationship Management (High Effort)

Challenge: Creating robust many-to-many relationships between vendors and events **Implementation Effort**:

- Designed PersonId system for stable relationship references
- Built assignment commands (assign/unassign) with bulk operations
- Implemented relationship integrity checks and validation
- Created participant tracking within events
- Developed UI components to display relationships

Complexity: Managing bidirectional relationships while maintaining data integrity required careful design and extensive testing.

3. Budget Management System (Medium-High Effort)

Challenge: Real-time budget tracking and validation across assignments **Implementation Effort**:

- Added Budget class with validation and arithmetic operations
- Implemented budget constraint checking in assignment commands
- Created remaining budget calculations for events
- Built budget visualization in UI components
- Added budget-aware error handling and user feedback

Critical Feature: Budget validation prevents overspending and is core to the party planning domain.

4. Advanced Command System (Medium Effort)

Challenge: Extending basic CRUD to domain-specific operations **Implementation Effort**:

- Built AddEventCommand with optional bulk assignment

- Created AssignContactToEventCommand with sophisticated validation
- Developed confirmation system for destructive operations (ConfirmClearCommand)
- Enhanced search to work across both names and tags
- Added undo system with state preservation

Domain Integration: Commands specifically designed for party planning workflows.

5. Enhanced User Interface (Medium Effort)

Challenge: Adapting UI for dual-entity display and specialized workflows **Implementation Effort**:

- Created EventListPanel with participant count display
- Enhanced PersonListPanel with budget and website information
- Developed budget status indicators and color coding
- Built responsive layout handling for both entity types
- Added comprehensive help system with scrollable reference

Technical Challenges Overcome

1. JSON Serialization Complexity

- **Challenge**: Serializing PersonId references and maintaining relationships
- **Solution**: Custom JSON adapters with careful ID preservation
- **Effort**: Multiple iterations to ensure data integrity across sessions

2. Observable List Management

- **Challenge**: Maintaining UI responsiveness with dual filtered lists
- **Solution**: Proper JavaFX binding and update mechanisms
- **Effort**: Required deep understanding of JavaFX observable patterns

3. Command Parser Extension

- **Challenge**: Adding complex command syntax while maintaining consistency
- **Solution**: Extended parser framework with new prefix types and validation
- **Effort**: Careful design to maintain backward compatibility

4. Budget Arithmetic Precision

- **Challenge**: Ensuring accurate financial calculations
- **Solution**: Robust number parsing and validation with proper error handling
- **Effort**: Multiple test scenarios to ensure calculation accuracy

Quantitative Metrics

- **Total Java Files**: 108 (significant increase from base project)
- **New Model Classes**: 15+ (Event hierarchy, PersonId, Budget)
- **New Commands**: 8 (AddEvent, AssignContact, UnassignContact, etc.)
- **New UI Components**: 5 (EventListPanel, enhanced displays)

- **Test Coverage**: Comprehensive unit and integration tests

Team Coordination Effort

- **Architecture Decisions**: Multiple team discussions on entity relationships
- **UI/UX Design**: Iterative refinement based on user feedback
- **Integration Testing**: Extensive testing of component interactions
- **Documentation**: Comprehensive developer and user guides

Comparison to AB3

Aspect	AddressBook-Level3	AbsolutSin-ema	Effort Multiplier
Core Entities	1 (Person)	2 (Person + Event)	2x
Relationships	None	Many-to-many	New
Domain Logic	Generic	Party Planning	3x
Budget System	None	Comprehensive	New
Command Complexity	Basic CRUD	Domain Operations	2x
UI Complexity	Single list	Dual entity display	1.5x

Conclusion

AbsolutSin-ema represents a substantial evolution from the base project, requiring significant architectural changes, new domain modeling, and specialized user experience design. The effort invested has resulted in a professional-grade application tailored specifically for party planning workflows, demonstrating the team's ability to transform a generic template into a domain-specific solution.

The project successfully balances feature richness with maintainability, providing a solid foundation for future enhancements while delivering immediate value to party planning professionals.