­­

Contents

[1. Introduction 5](#_Toc510389889)

[2. Setting up 5](#_Toc510389890)

[2.1. Prerequisites 5](#_Toc510389891)

[2.2. Setting up the project in your computer 6](#_Toc510389892)

[2.3. Verifying the setup 6](#_Toc510389893)

[2.4. Configurations to do before writing code 7](#_Toc510389894)

[2.4.1. Configuring the coding style 7](#_Toc510389895)

[2.4.2. Updating documentation to match your fork 7](#_Toc510389896)

[2.4.3. Setting up CI 8](#_Toc510389897)

[2.4.4. Getting started with coding 8](#_Toc510389898)

[3. Design 8](#_Toc510389899)

[3.1. Architecture 9](#_Toc510389900)

[3.2. UI component 12](#_Toc510389901)

[3.3. Logic component 14](#_Toc510389902)

[3.4. Model component 15](#_Toc510389903)

[3.5. Storage component 16](#_Toc510389904)

[3.6. Common classes 17](#_Toc510389905)

[4. Implementation 17](#_Toc510389906)

[4.1. Undo/Redo feature 17](#_Toc510389907)

[4.1.1. Current Implementation 18](#_Toc510389908)

[4.1.2. Design Considerations 22](#_Toc510389909)

[4.2. Add ToDo feature 25](#_Toc510389910)

[4.2.1. Current Implementation 25](#_Toc510389911)

[4.2.2. Design Considerations 27](#_Toc510389912)

[4.3. Check/Uncheck ToDo feature 28](#_Toc510389913)

[4.3.1. Current Implementation 28](#_Toc510389914)

[4.3.2. Design Considerations 29](#_Toc510389915)

[4.4. Logging 30](#_Toc510389916)

[4.5. Configuration 31](#_Toc510389917)

[4.6. NUSmods Integration 31](#_Toc510389918)

[4.6.1. Current Implementation 31](#_Toc510389919)

[4.6.2. Design Considerations 31](#_Toc510389920)

[4.7. List Group Members Feature 32](#_Toc510389921)

[4.7.1. Current Implementation 32](#_Toc510389922)

[4.7.2. Design Considerations 33](#_Toc510389923)

[4.8. Calendar Feature 33](#_Toc510389924)

[4.8.1. Current Implementation 33](#_Toc510389925)

[4.8.2. Design Considerations 34](#_Toc510389926)

[4.9. Add Group feature 35](#_Toc510389927)

[4.9.1. Current Implementation 35](#_Toc510389928)

[4.9.2. Design Considerations 37](#_Toc510389929)

[5. Documentation 38](#_Toc510389930)

[5.1. Editing Documentation 38](#_Toc510389931)

[5.2. Publishing Documentation 38](#_Toc510389932)

[5.3. Converting Documentation to PDF format 39](#_Toc510389933)

[6. Testing 40](#_Toc510389934)

[6.1. Running Tests 40](#_Toc510389935)

[6.2. Types of tests 41](#_Toc510389936)

[6.3. Troubleshooting Testing 42](#_Toc510389937)

[7. Dev Ops 42](#_Toc510389938)

[7.1. Build Automation 42](#_Toc510389939)

[7.2. Continuous Integration 43](#_Toc510389940)

[7.3. Coverage Reporting 43](#_Toc510389941)

[7.4. Documentation Previews 43](#_Toc510389942)

[7.5. Making a Release 43](#_Toc510389943)

[7.6. Managing Dependencies 44](#_Toc510389944)

[Appendix A: Product Scope 44](#_Toc510389945)

[Appendix B: User Stories 46](#_Toc510389946)

[Appendix C: Use Cases 51](#_Toc510389947)

[Appendix D: Non Functional Requirements 55](#_Toc510389948)

[Appendix E: Glossary 56](#_Toc510389949)

[Appendix F: Instructions for Manual Testing 56](#_Toc510389950)

[F.1. Launch and Shutdown 57](#_Toc510389951)

[F.2. Deleting a person 57](#_Toc510389952)

[F.3. Saving data 58](#_Toc510389953)

# Introduction

Fastis is a desktop application that aims to help students who are studying at National University of Singapore (NUS) manage their group projects effectively. The application enables students to keep track of group members’ timetables, information, meetups agendas, to-dos, and schedule suitable meetup time.

This Developer Guide is meant to provides the necessary information for developers who are interested in contributing to this application. The guide consists of the set-up steps to develop the application and the detailed descriptions and implementations of the application’s current features.

# Setting up

This section provides information on setting up the project in order to work on Fastis.

## Prerequisites

The following applications should be present before setting up:

JDK **1.8.0\_60** or later

|  |  |
| --- | --- |
| ℹ️ | Having any Java 8 version is not enough.  This app will not work with earlier versions of Java 8. |

IntelliJ IDE

|  |  |
| --- | --- |
| ℹ️ | IntelliJ by default has Gradle and JavaFx plugins installed.  Do not disable them. If you have disabled them, go to File > Settings > Plugins to re-enable them. |

## Setting up the project in your computer

To set up the project, follow these steps:

1. Fork this repo, and clone the fork to your computer
2. Open IntelliJ (if you are not in the welcome screen, click File > Close Project to close the existing project dialog first)
3. Set up the correct JDK version for Gradle
4. Click Configure > Project Defaults > Project Structure
5. Click New…​ and find the directory of the JDK
6. Click Import Project
7. Locate the build.gradle file and select it. Click OK
8. Click Open as Project
9. Click OK to accept the default settings
10. Open a console and run the command gradlew processResources (Mac/Linux: ./gradlew processResources). It should finish with the BUILD SUCCESSFUL message.
11. This will generate all resources required by the application and tests.

## Verifying the setup

The following steps should be done to verify that the setup of Fastis is correct.

1. Run the seedu.address.MainApp and try a few commands
2. Run the tests to ensure they all pass.

## Configurations to do before writing code

The following configurations should be done to ensure that Fastis future development follows good coding standards and practices.

### Configuring the coding style

This project follows [oss-generic coding standards](https://github.com/oss-generic/process/blob/master/docs/CodingStandards.adoc). IntelliJ’s default style is mostly compliant with ours but it uses a different import order from ours. To rectify,

1. Go to File > Settings…​ (Windows/Linux), or IntelliJ IDEA > Preferences…​ (macOS)
2. Select Editor > Code Style > Java
3. Click on the Imports tab to set the order

* For Class count to use import with '\*' and Names count to use static import with '\*': Set to 999 to prevent IntelliJ from contracting the import statements
* For Import Layout: The order is import static all other imports, import java.\*, import javax.\*, import org.\*, import com.\*, import all other imports. Add a <blank line> between each import

Optionally, you can follow the [UsingCheckstyle.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingCheckstyle.adoc) document to configure Intellij to check style-compliance as you write code.

### Updating documentation to match your fork

After forking the repo, links in the documentation will still point to the CS2103JAN2018-W15-B3/main repo. If you plan to develop this as a separate product (i.e. instead of contributing to the CS2103JAN2018-W15-B3/main), you should replace the URL in the variable repoURL in DeveloperGuide.adoc and UserGuide.adoc with the URL of your fork.

### Setting up CI

You should set up Travis to perform Continuous Integration (CI) for your fork. See [UsingTravis.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingTravis.adoc) to learn how to set it up.

After setting up Travis, you can optionally set up coverage reporting for your team fork (see [UsingCoveralls.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingCoveralls.adoc)).

|  |  |
| --- | --- |
| ℹ️ | Coverage reporting could be useful for a team repository that hosts the final version but it is not that useful for your personal fork. |

Optionally, you can set up AppVeyor as a second CI (see [UsingAppVeyor.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingAppVeyor.adoc)).

|  |  |
| --- | --- |
| ℹ️ | Having both Travis and AppVeyor ensures your App works on both Unix-based platforms and Windows-based platforms (Travis is Unix-based and AppVeyor is Windows-based) |

### Getting started with coding

When you are ready to start coding, get some sense of the overall design by reading [Section 3.1, “Architecture”](#_Architecture).

# Design

The following sections describes how Fastis is built and how its different components interact and work with each other.

## Architecture

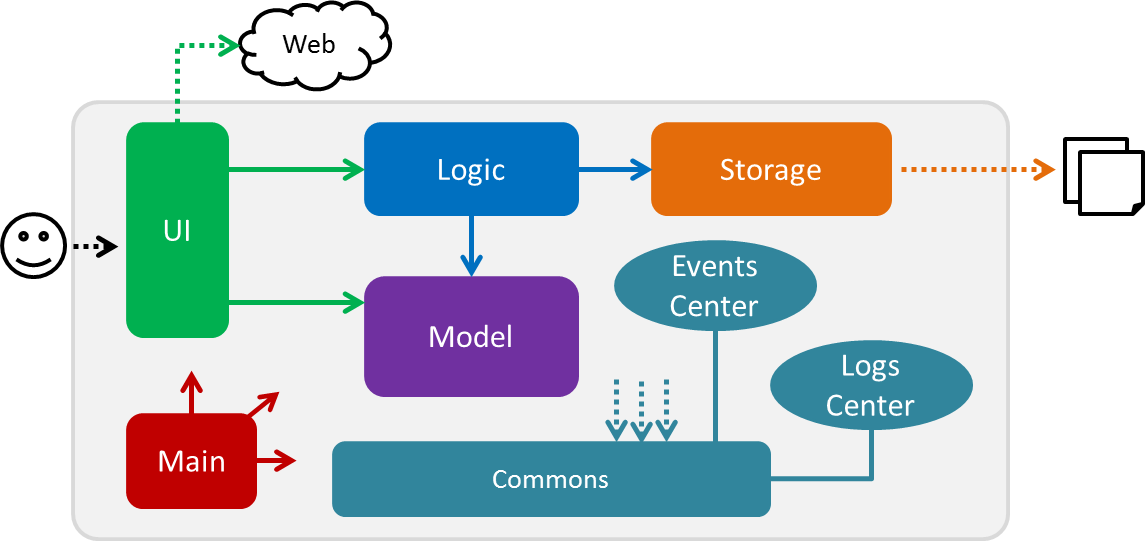
The *Architecture Diagram* given below (Figure 1) explains the high-level design of Fastis. Given below is a quick overview of each component.

Figure 1. Architecture Diagram

|  |  |
| --- | --- |
| 💡 | The .pptx files used to create diagrams in this document can be found in the [diagrams](https://github.com/CS2103JAN2018-W15-B3/main/docs/diagrams/) folder. To update a diagram, modify the diagram in the pptx file, select the objects of the diagram, and choose Save as picture. |

Main has only one class called [MainApp](https://github.com/CS2103JAN2018-W15-B3/main/src/main/java/seedu/address/MainApp.java). It is responsible for：

* At app launch: Initializes the components in the correct sequence, and connects them up with each other.
* At shut down: Shuts down the components and invokes cleanup method where necessary.

[Commons](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/DeveloperGuide.adoc#Design-Commons) represents a collection of classes used by multiple other components. Two of those classes play important roles at the architecture level.

* EventsCenter : This class (written using [Google’s Event Bus library](https://github.com/google/guava/wiki/EventBusExplained)) is used by components to communicate with other components using events (i.e. a form of Event Driven design)
* LogsCenter : Used by many classes to write log messages to the App’s log file.

The rest of the App consists of four components:

[UI](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/DeveloperGuide.adoc#Design-Ui): The UI of the App.

[Logic](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/DeveloperGuide.adoc#Design-Logic): The command executor.

[Model](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/DeveloperGuide.adoc#Design-Model): Holds the data of the App in-memory.

[Storage](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/DeveloperGuide.adoc#Design-Storage): Reads data from, and writes data to, the hard disk.

Each of the four components:

* Defines its API in an interface with the same name as the Component.
* Exposes its functionality using a {Component Name}Manager class.

For example, the Logic component (see the class diagram ,Figure 2, given below) defines it’s API in the Logic.java interface and exposes its functionality using the LogicManager.java class.

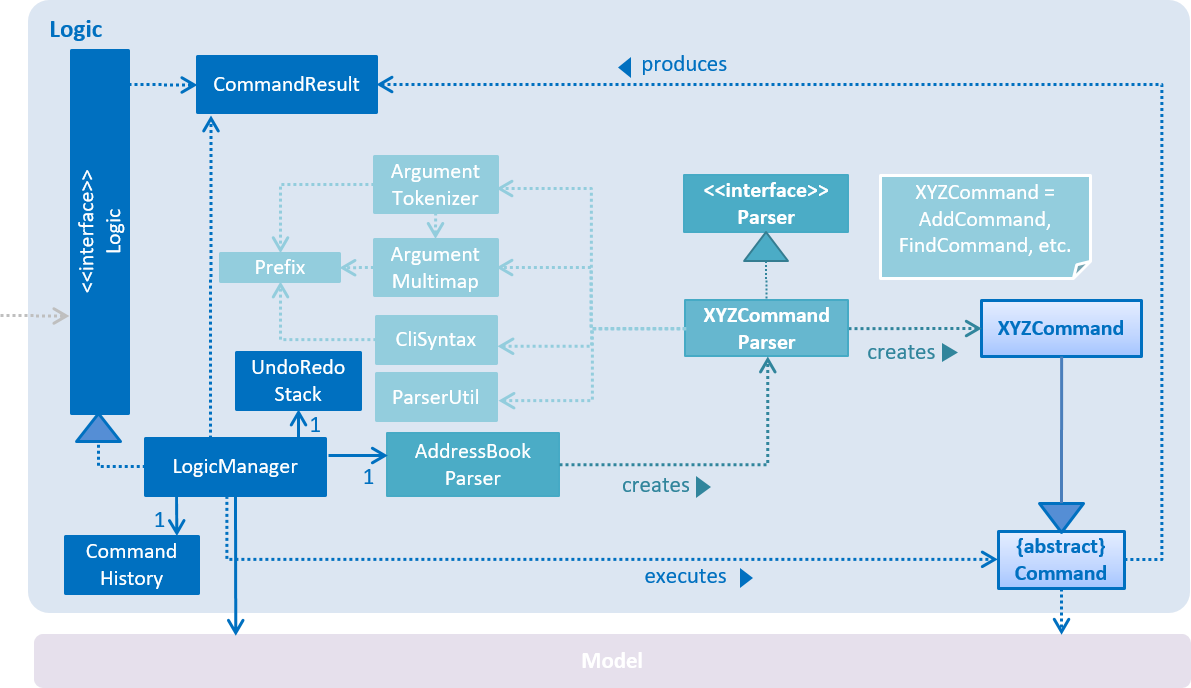


Figure 2. Class Diagram of the Logic Component

**Events-Driven nature of the design**

The *Sequence Diagram* in shown in Figure 3 shows how the components interact for the scenario where the user issues the command delete 1.

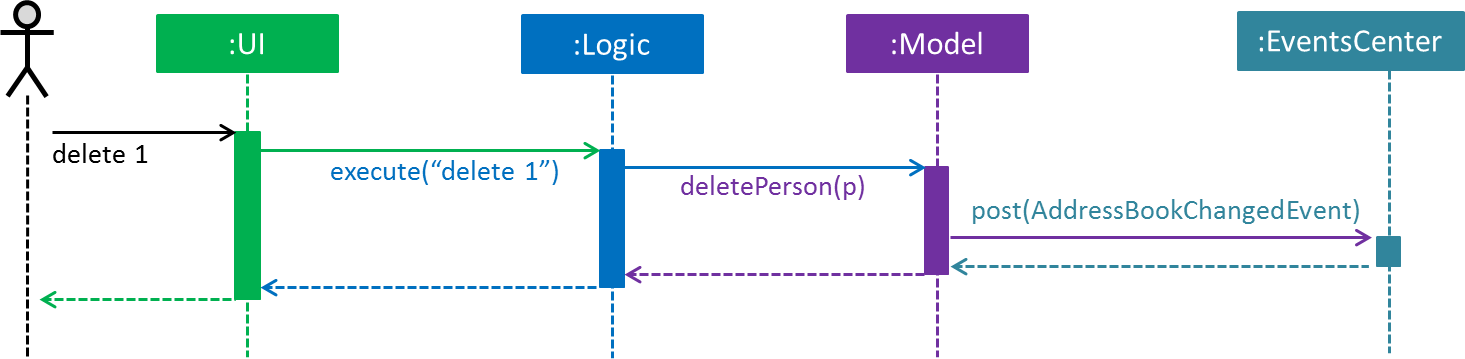


Figure 3. Component interactions for delete 1 command (part 1)

|  |  |
| --- | --- |
| ℹ️ | Note how the Model simply raises a AddressBookChangedEvent when the Address Book data are changed, instead of asking the Storage to save the updates to the hard disk. |

Figure 4 below shows how the EventsCenter reacts to that event, which eventually results in the updates being saved to the hard disk and the status bar of the UI being updated to reflect the 'Last Updated' time.

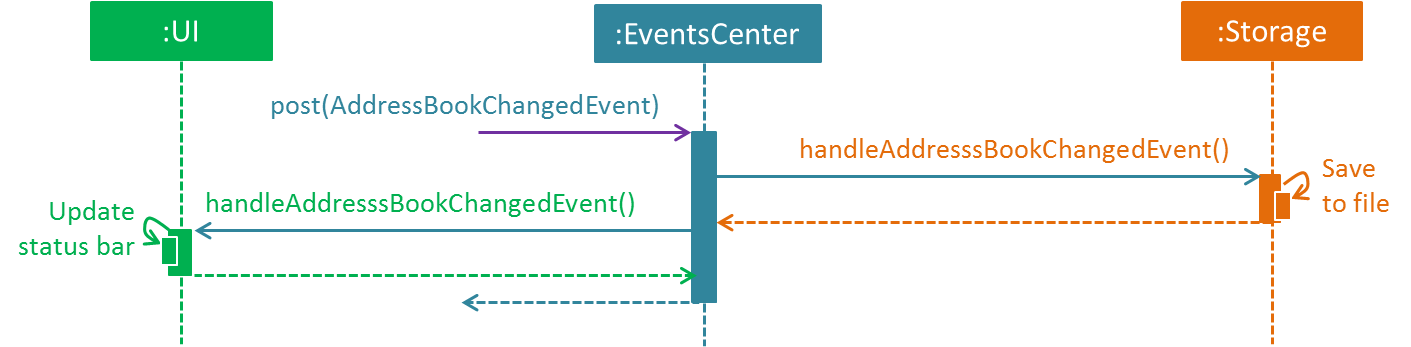


Figure 4. Component interactions for delete 1 command (part 2)

|  |  |
| --- | --- |
| ℹ️ | Note how the event is propagated through the EventsCenter to the Storage and UI without Model having to be coupled to either of them. This is an example of how this Event Driven approach helps us reduce direct coupling between components. |

The sections below give more details of each component.

## UI component

The UI component handles the inputs from and the outputs to the User Interface. It consists of a MainWindow that is made up of parts e.g. CommandBox, ResultDisplay, PersonListPanel, StatusBarFooter, BrowserPanel etc. All these, including the MainWindow, inherit from the abstract UiPart class. Refer to Figure 5 for the structure of the UI Component.

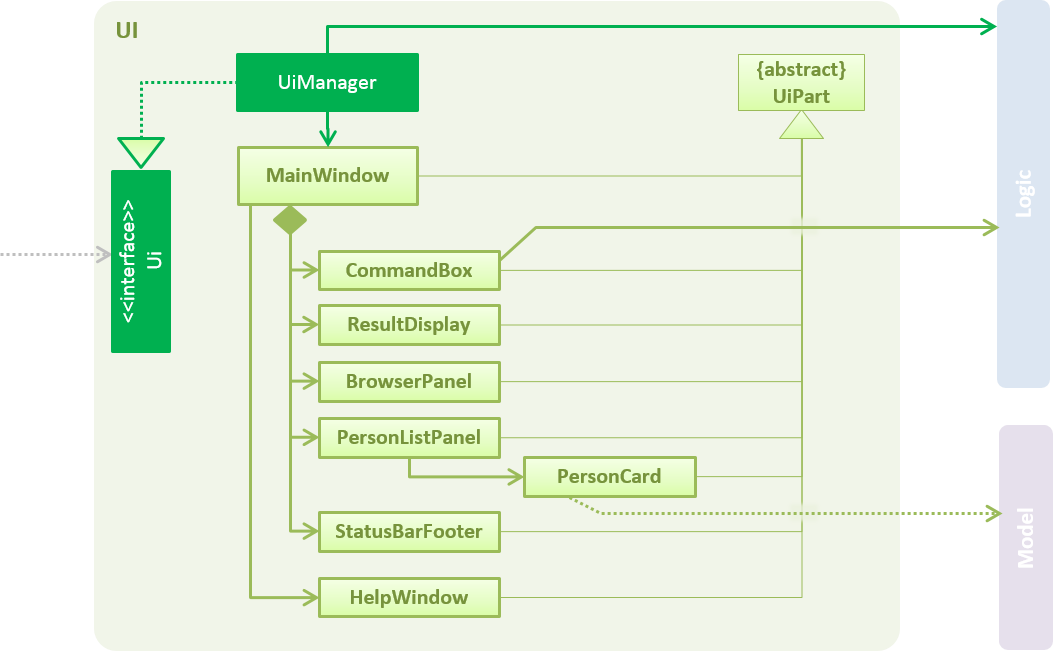


Figure 5. Structure of the UI Component

API : [Ui.java](https://github.com/CS2103JAN2018-W15-B3/main/src/main/java/seedu/address/ui/Ui.java)

The UI component uses JavaFx UI framework. The layout of these UI parts are defined in matching .fxml files that are in the src/main/resources/view folder. For example, the layout of the [MainWindow](https://github.com/CS2103JAN2018-W15-B3/main/src/main/java/seedu/address/ui/MainWindow.java) is specified in [MainWindow.fxml](https://github.com/CS2103JAN2018-W15-B3/main/src/main/resources/view/MainWindow.fxml)

The UI component:

* Executes user commands using the Logic component.
* Binds itself to some data in the Model so that the UI can auto-update when data in the Model change.
* Responds to events raised from various parts of the App and updates the UI accordingly.

## Logic component

The Logic component handles the commands from user and passes the command results to the User Interface.

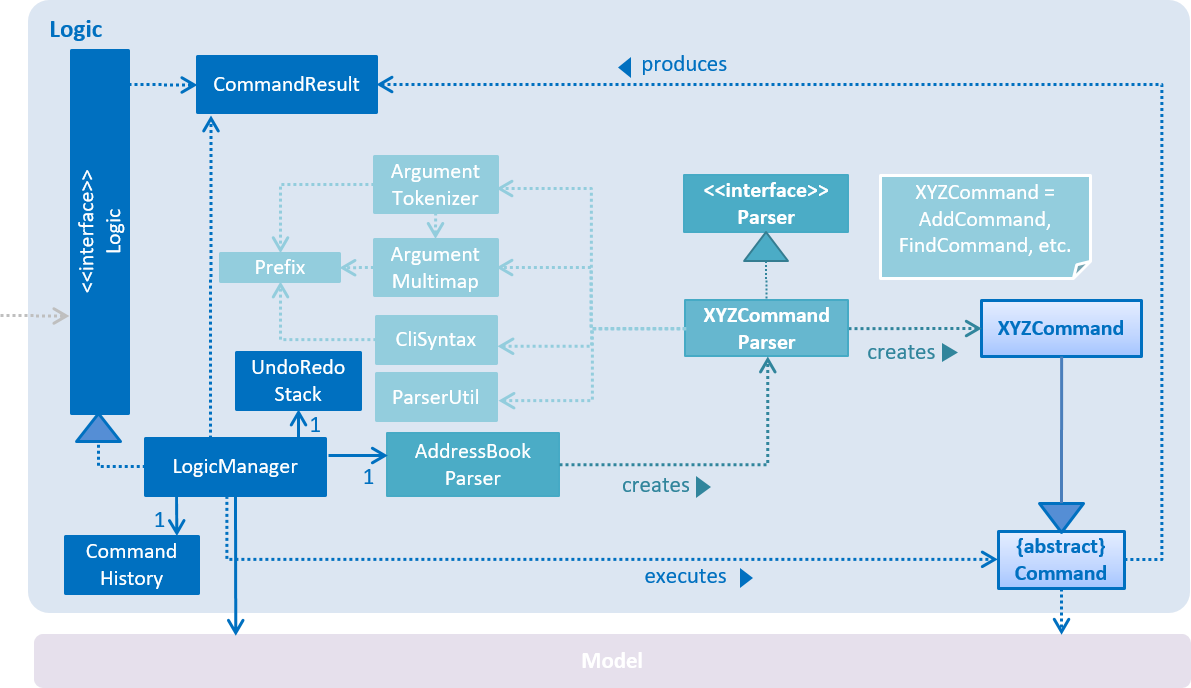


Figure 6. Structure of the Logic Component

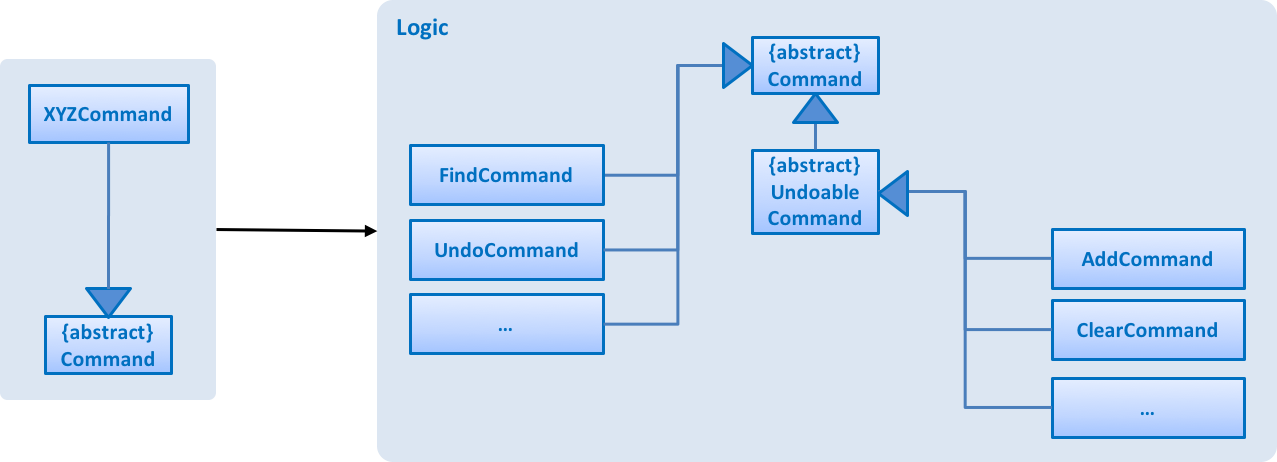


Figure 7. Structure of Commands in the Logic Component. This diagram shows finer details concerning XYZCommand and Command in [Figure 6, “Structure of the Logic Component”](#fig6)

API : [Logic.java](https://github.com/CS2103JAN2018-W15-B3/main/src/main/java/seedu/address/logic/Logic.java)

Given below is the workflow of the Logic Component when the user inputs a command:

1. Logic uses the AddressBookParser class to parse the user command into a command object.
2. This Command object is then executed by the LogicManager.
3. Depending on the command, execution might alter the Model (e.g. adding a person) and/or raise events.
4. The result and user feedback from the command execution is encapsulated as a CommandResult object which is passed back to the UI.

Figure 8 shows the Sequence Diagram for an example interaction within the Logic component for the execute("delete 1") API call.

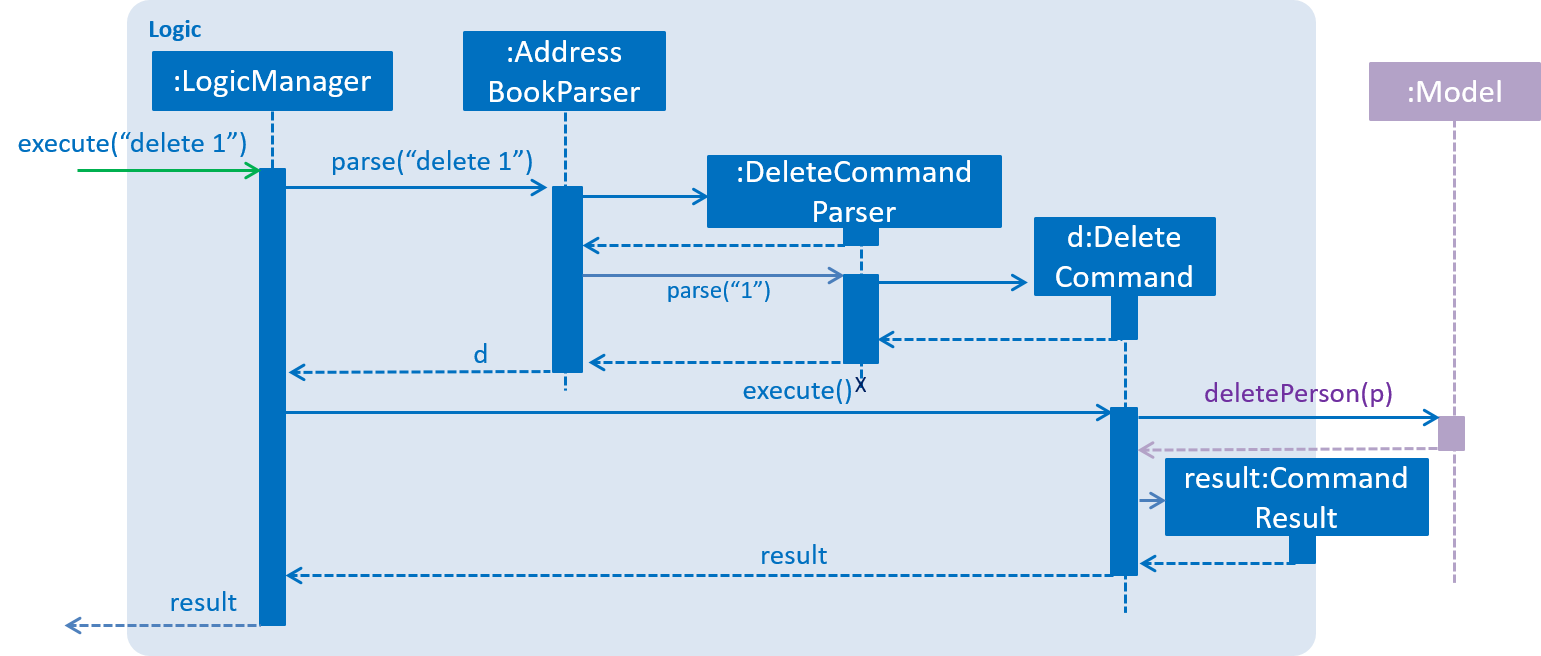


Figure 8. Interactions Inside the Logic Component for the delete 1 Command

## Model component

The model component stores and operates on the data held by Fastis as shown in Figure 9.

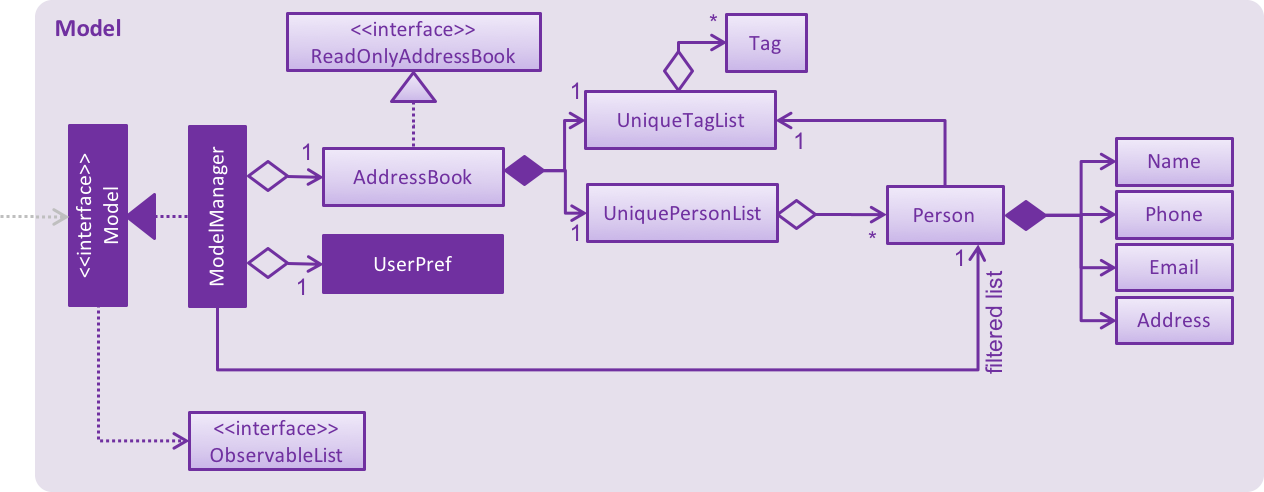


Figure 9. Structure of the Model Component

API : [Model.java](https://github.com/CS2103JAN2018-W15-B3/main/src/main/java/seedu/address/model/Model.java)

The Model:

* stores a UserPref object that represents the user’s preferences.
* stores the Address Book data.
* exposes an unmodifiable ObservableList<Person> that can be 'observed' e.g. the UI can be bound to this list so that the UI automatically updates when the data in the list change.

The model component does not depend on any of the other three components, meaning that it does not rely on any functions outside of itself to operate.

## Storage component

The storage component saves processed data from Fastis on to the running machine’s hard-disk and reads from the stored data as shown in Figure 10.

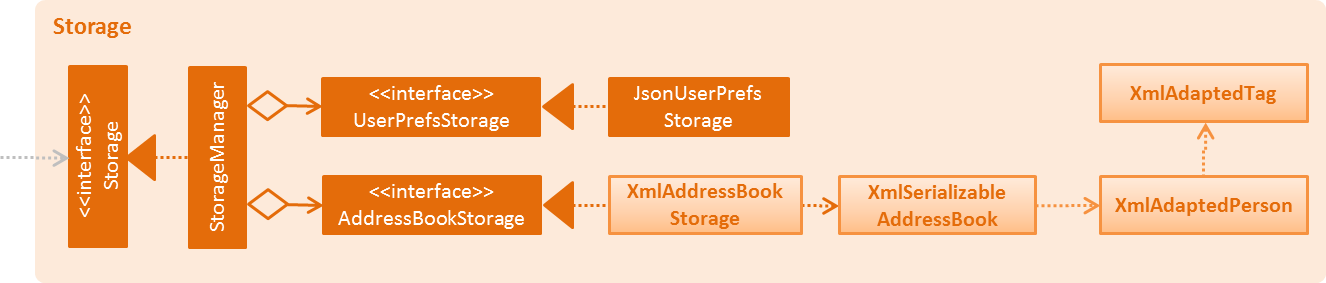


Figure 10. Structure of the Storage Component

API : [Storage.java](https://github.com/CS2103JAN2018-W15-B3/main/src/main/java/seedu/address/storage/Storage.java)

The Storage component is in charge of:

* saving UserPref objects in json format and reading it back on next startup.
* saving the Address Book data in xml format and reading it back on next startup.

## Common classes

Classes used by multiple components are in the seedu.addressbook.commons package.

# Implementation

This section describes some noteworthy details on how certain features are implemented.

## Undo/Redo feature

The Undo feature allows users to restore the state before the latest command while the Redo feature reverses the Undo command.

### Current Implementation

The undo/redo mechanism is facilitated by an UndoRedoStack, which holds the states of the model. The stack resides inside LogicManager. It supports undoing and redoing of commands that modifies the state of the address book (e.g. add, edit). Such commands will inherit from UndoableCommand.

UndoRedoStack only deals with UndoableCommands. Commands that cannot be undone will inherit from Command instead. Figure 11 shows the inheritance diagram for commands:

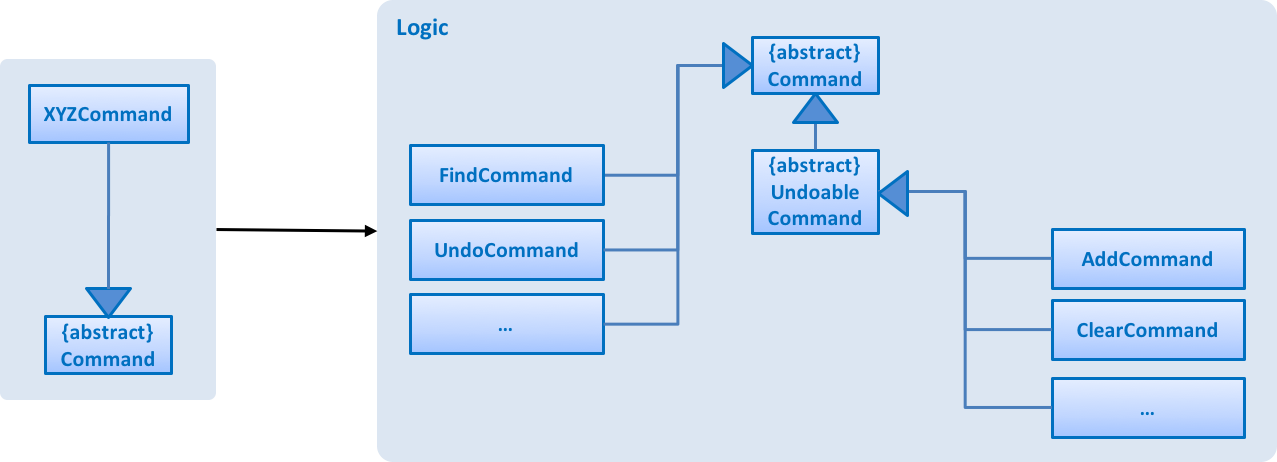


Figure 11. Class diagram Command classes

As seen from figure 11, UndoableCommand adds an extra layer between the abstract Command class and concrete commands that can be undone, such as the DeleteCommand. Note that extra tasks need to be done when executing a command in an *undoable* way, such as saving the state of the address book before execution. UndoableCommand contains the high-level algorithm for those extra tasks while the child classes implements the details of how to execute the specific command. Note that this technique of putting the high-level algorithm in the parent class and lower-level steps of the algorithm in child classes is also known as the [template pattern](https://www.tutorialspoint.com/design_pattern/template_pattern.htm).

Commands that are not undoable are implemented this way:

public class ListCommand extends Command {

@Override

public CommandResult execute() {

// ... list logic ...

}

}

With the extra layer, the commands that are undoable are implemented this way:

public abstract class UndoableCommand extends Command {

@Override

public CommandResult execute() {

// ... undo logic ...

executeUndoableCommand();

}

}

public class DeleteCommand extends UndoableCommand {

@Override

public CommandResult executeUndoableCommand() {

// ... delete logic ...

}

}

Suppose that the user has just launched the application. The UndoRedoStack will be empty at the beginning.

The user executes a new UndoableCommand, delete 5, to delete the 5th person in the address book. The current state of the address book is saved before the delete 5 command executes. The delete 5 command will then be pushed onto the undoStack (the current state is saved together with the command)(refer to Figure 12).



Figure 12. Delete’s action on stack

As the user continues to use the program, more commands are added into the undoStack(Figure 13). For example, the user may execute add n/David …​ to add a new person.

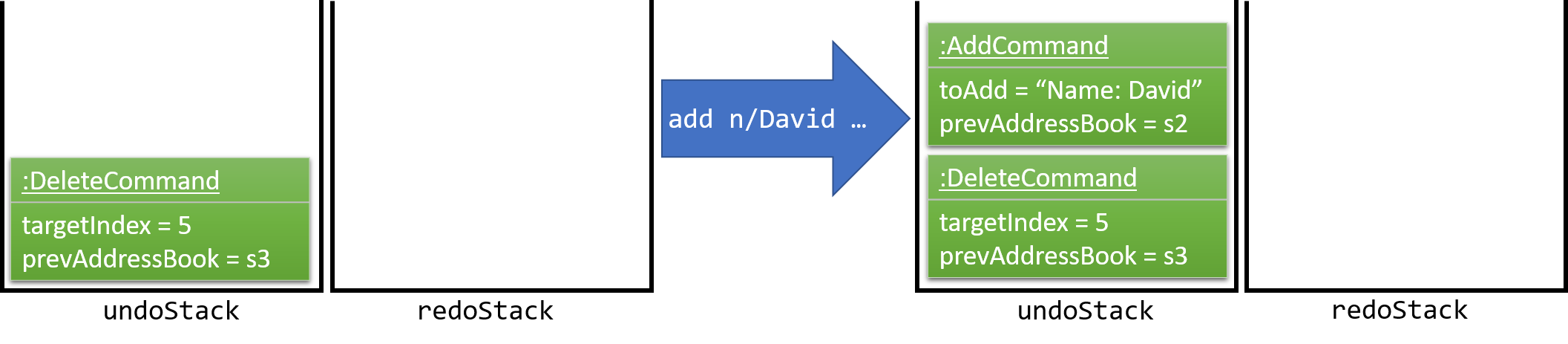


Figure 13. Commands added to undoStack.

|  |  |
| --- | --- |
| ℹ️ | If a command fails its execution, it will not be pushed to the UndoRedoStack at all. |

The user now decides that adding the person was a mistake, and decides to undo that action using undo.

Fastis then pops the most recent command out of the undoStack and pushes it onto the redoStack. Using the model state that is saved with the command, Fastis will restore the address book to the state before the add command executed(Figure 14).

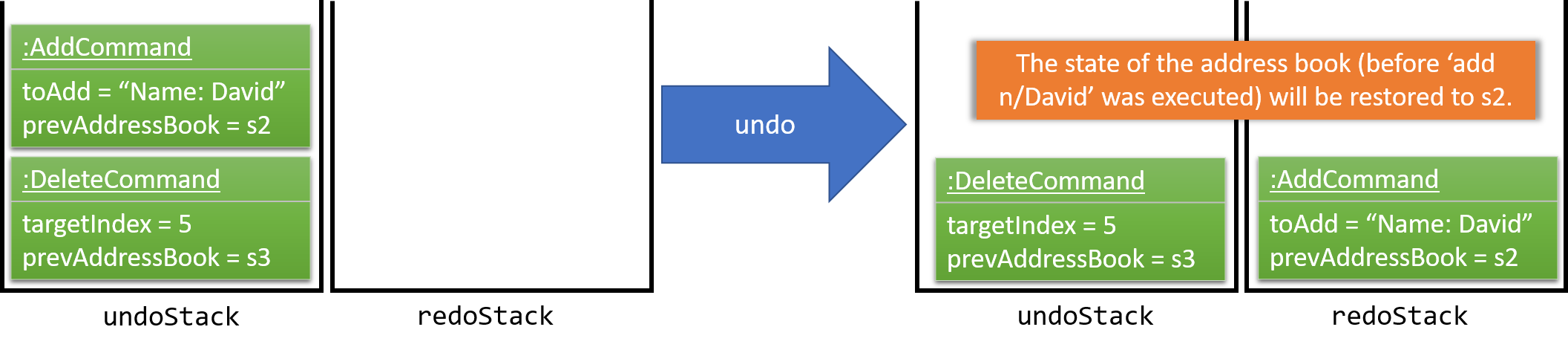


Figure 14. Return to state.

|  |  |
| --- | --- |
| ℹ️ | If the undoStack is empty, then there are no other commands left to be undone, and an Exception will be thrown when popping the undoStack. |

The following sequence diagram (Figure 15) shows how the undo operation works:

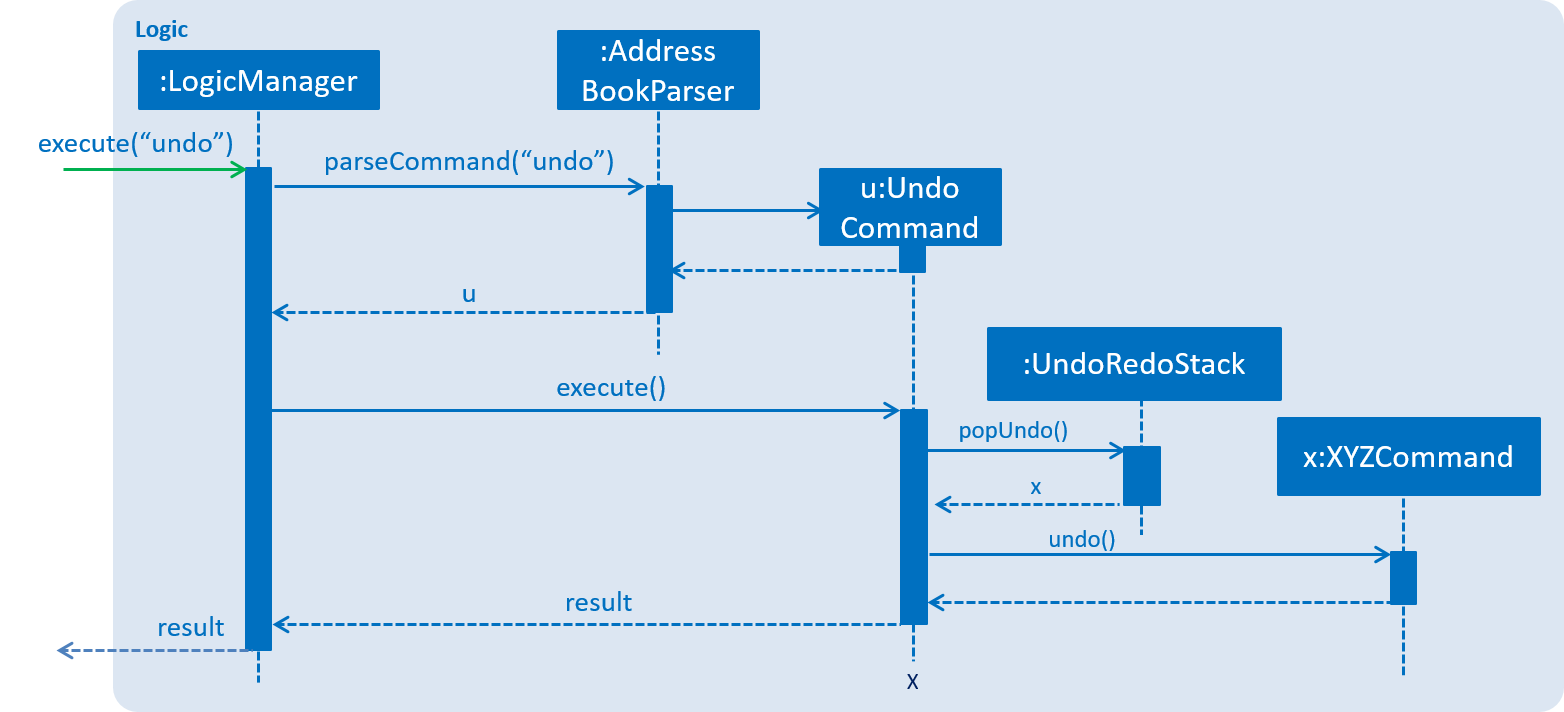


Figure 15. Interactions Inside the Logic Component for the Undo Command

The redo does the exact opposite (pops from redoStack, push to undoStack, and restores the address book to the state after the command is executed).

|  |  |
| --- | --- |
| ℹ️ | If the redoStack is empty, then there are no other commands left to be redone, and an Exception will be thrown when popping the redoStack. |

The user now decides to execute a new command, clear. As before, clear will be pushed into the undoStack(Figure 16). This time the redoStack is no longer empty. It will be purged as it no longer makes sense to redo the add n/David command (this is the behavior that most modern desktop applications follow).

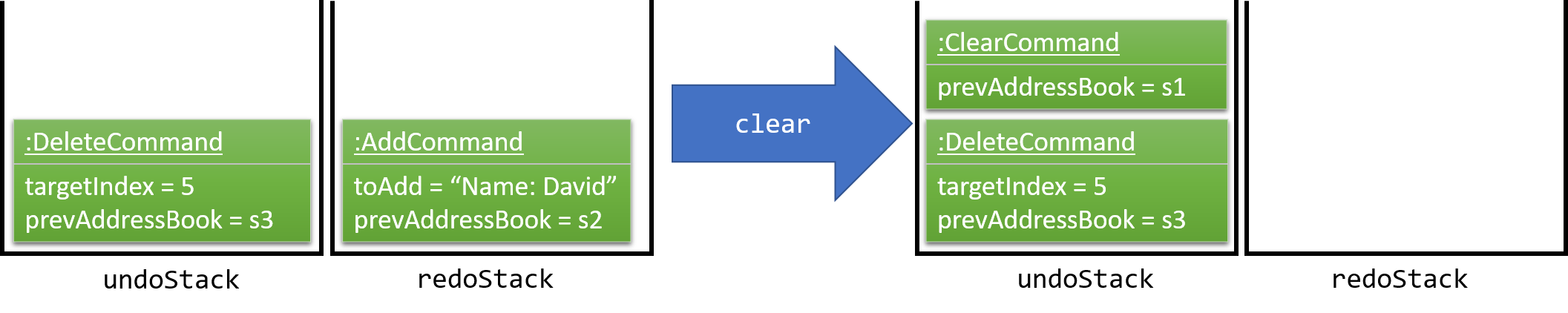


Figure 16. Clear command pushed into undoStack.

Commands that are not undoable are not added into the undoStack. For example, list, which inherits from Command rather than UndoableCommand, will not be added after execution(as shown in Figure 17):

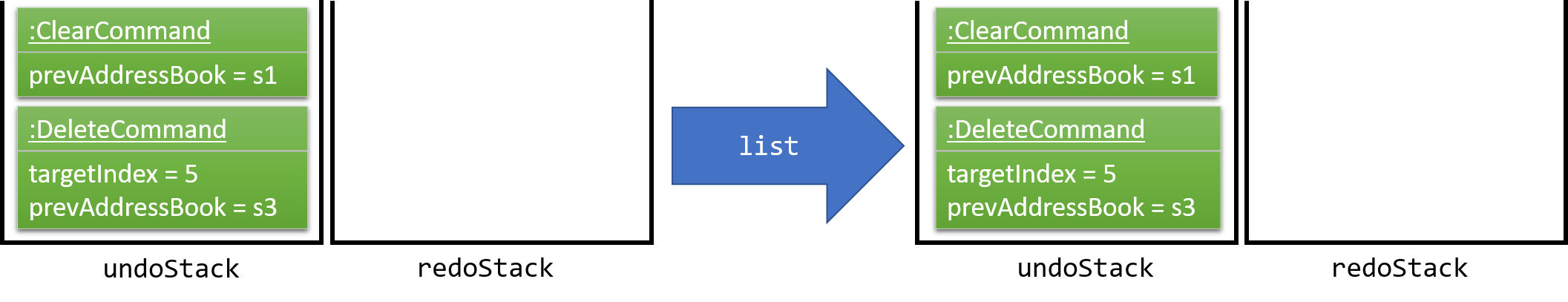


Figure 17. List is not added into the undoStack.

The following activity diagram (Figure 18) summarize what happens inside the UndoRedoStack when a user executes a new command:

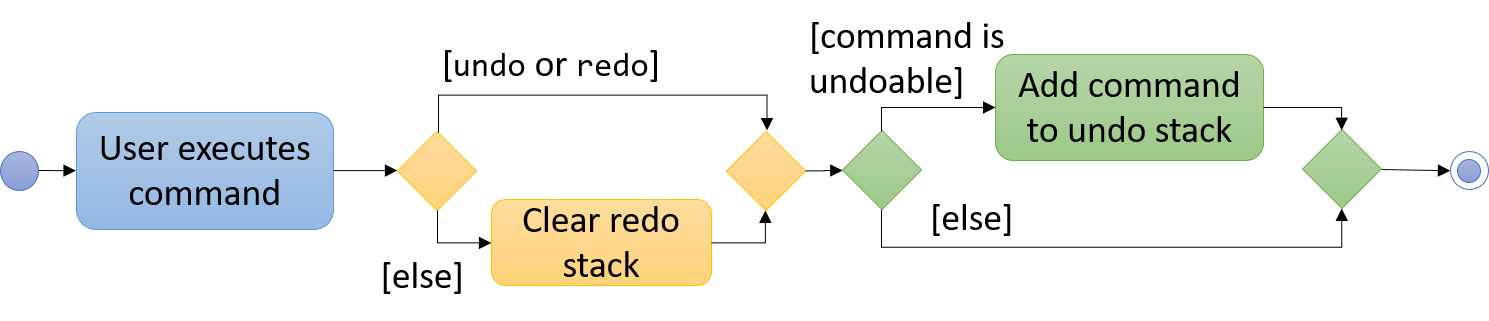


Figure 18. The Undo/Redo Activity Diagram

### Design Considerations

The following considerations were taken into account during the design of this feature.

Aspect: Implementation of UndoableCommand

* Alternative 1 (current choice): Adds a new abstract method executeUndoableCommand()
  + Pros: Fastis will not lose any undone/redone functionality as it is now part of the default behaviour. Classes that deal with Command do not have to know that executeUndoableCommand() exist.
  + Cons: New developers will find it hard to understand the template pattern.
* Alternative 2: Overrides execute()
  + Pros: This implementation does not involve the template pattern, which is easier for new developers to understand.
  + Cons: Classes that inherit from UndoableCommand must remember to call super.execute(), or lose the ability to undo/redo.

Aspect: How undo & redo executes

* Alternative 1 (current choice): Saves the entire address book.
  + Pros: This implementation is easy to implement.
  + Cons: This implementation may have performance issues in terms of memory usage.
* Alternative 2: Enable individual command to know how to undo/redo by itself.
  + Pros: This implementation uses less memory (e.g. for delete, just save the person being deleted).
  + Cons: Developers must ensure that the implementation of each individual command are correct.

Aspect: Type of commands that can be undone/redone

* Alternative 1 (current choice): Only includes commands that modifies the address book (add, clear, edit).
  + Pros: Fastis only revert changes that are hard to change back (the view can easily be re-modified as no data are \* lost).
  + Cons: Users might think that undo also applies when the list is modified (e.g. undoing filtering), only to realize that it does not do that, after executing undo.
* Alternative 2: Includes all commands.
  + Pros: Fastis might be more intuitive for users to use.
  + Cons: Users have no way of skipping such commands if they just want to reset the state of the address \* book and not the view.

Aspect: Data structure to support the undo/redo commands

* Alternative 1 (current choice): Uses separate stack for undo and redo
  + Pros: Easy to understand for new Computer Science student undergraduates to understand, who are likely to be \* the new incoming developers of our project.
  + Cons: Logic is duplicated twice. For example, when a new command is executed, we must remember to update \* both HistoryManager and UndoRedoStack.
* Alternative 2: Use HistoryManager for undo/redo
  + Pros: We do not need to maintain a separate stack, and just reuse what is already in the codebase.
  + Cons: Requires dealing with commands that have already been undone: We must remember to skip these commands. Violates Single Responsibility Principle and Separation of Concerns as HistoryManager now needs to do two \* different things.

## Add ToDo feature

The Add ToDo feature adds a to-do to the to-do list for users to keep track of tasks to do in group projects.

### Current Implementation

The add to-dos mechanism is facilitated by AddToDoCommand, which resides inside Logic component. It supports adding ToDo objects to the address book. AddToDoCommand inherits from UndoableCommand.

Hence, AddToDoCommand can be undone using UndoRedoStack. With the extra layer, the undoable AddToDoCommand is implemented this way:

public abstract class UndoableCommand extends Command {

@Override

public CommandResult execute() {

// ... undo logic ...

executeUndoableCommand();

}

}

public class AddToDoCommand extends UndoableCommand {

@Override

public CommandResult executeUndoableCommand() {

// ... delete logic ...

}

}

The to-dos in the to-do list are facilitated by ToDo class. Each ToDo object has a Content object, representing the content of the to-do. Address book stores all to-dos in UniqueToDoList. ToDo, Content and UniqueToDoList classes reside inside Model component. Figure 19 shows the interactions between ToDo, Content and UniqueToDoList:

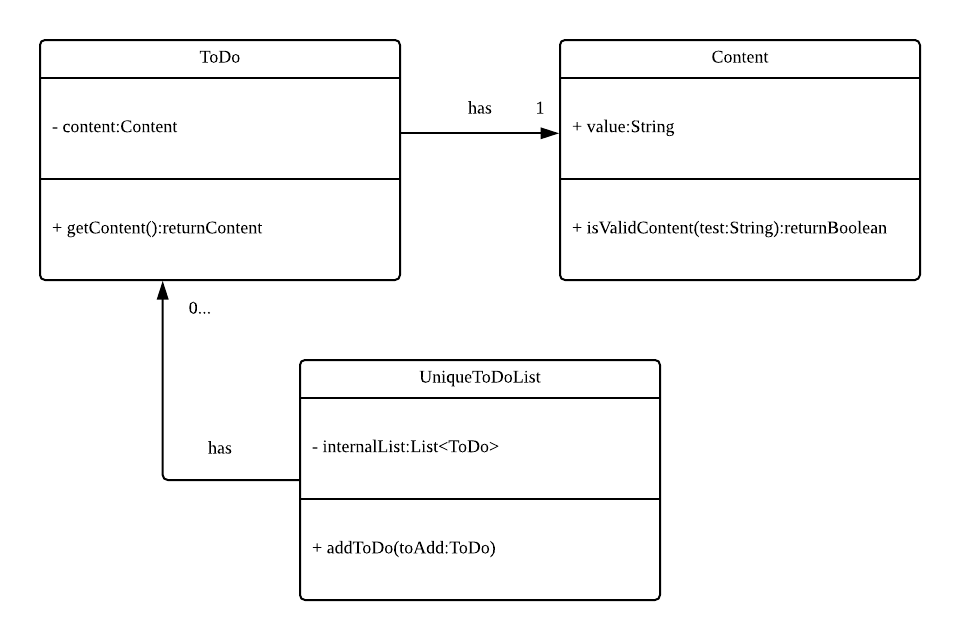


Figure 19. Interactions between Content, ToDo and UniqueToDoList classes

Suppose that the user has just launched the application. The UniqueToDoList in the address book will be empty if no to-dos have been added previously.

The user executes a new AddToDoCommand with Content, to add a new to-do to the address book. The new to-do is added to the UniqueToDoList and the current state of the address book is saved. Figure 20 shows the interactions the Logic component when the addToDo Command is called:

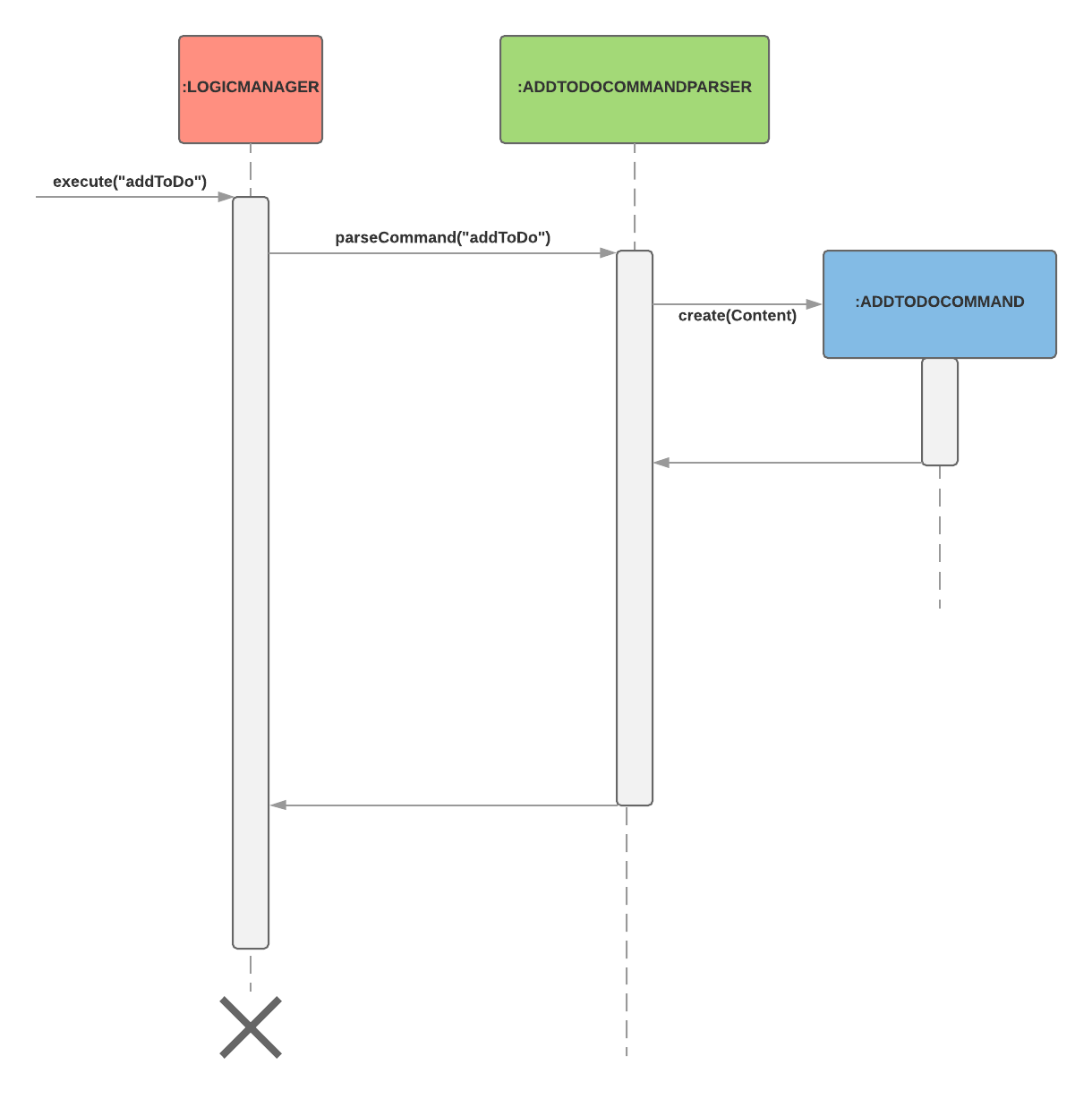


Figure 20. Interactions inside the Logic component for addToDo Command

### Design Considerations

Aspect: Implementation of AddToDoCommand

* Alternative 1 (current choice): Adds a new abstract method executeAddToDoCommand()
  + Pros: Fastis will not lose any addToDo functionality as it is now part of the default behavior. Classes that deal with AddToDoCommand do not have to know that executeAddToDoCommand() exist.
  + Cons: New developers will find it hard to understand the template pattern.
* Alternative 2: Just override execute()
  + Pros: Fatis does not involve the template pattern, easier for new developers to understand.
  + Cons: Classes that inherit from AddToDoCommand must remember to call super.execute(), or lose the ability to addToDo.

## Check/Uncheck ToDo feature

### Current Implementation

The check/uncheck to-dos mechanism is facilitated by CheckToDoCommand and UnCheckToDoCommand, which resides inside Logic component. It supports modifying Status objects within ToDo objects. CheckToDoCommand and UnCheckToDoCommand inherit from UndoableCommand.

Hence, CheckToDoCommand and UnCheckToDoCommand can be undone using UndoRedoStack. With the extra layer, the CheckToDoCommand and UnCheckToDoCommand that are undoable are implemented this way:

public abstract class UndoableCommand extends Command {

@Override

public CommandResult execute() {

// ... undo logic ...

executeUndoableCommand();

}

}

public class CheckToDoCommand extends UndoableCommand {

@Override

public CommandResult executeUndoableCommand() {

// ... check to-do logic ...

}

}

public class UnCheckToDoCommand extends UndoableCommand {

@Override

public CommandResult executeUndoableCommand() {

// ... uncheck to-do logic ...

}

}

Similar to Content object, each ToDo object have a Status object, representing the status of the to-do. The status of a to-do can be either done or undone. Status class resides inside Model component. Figure 21 shows the interaction between ToDo and Status:

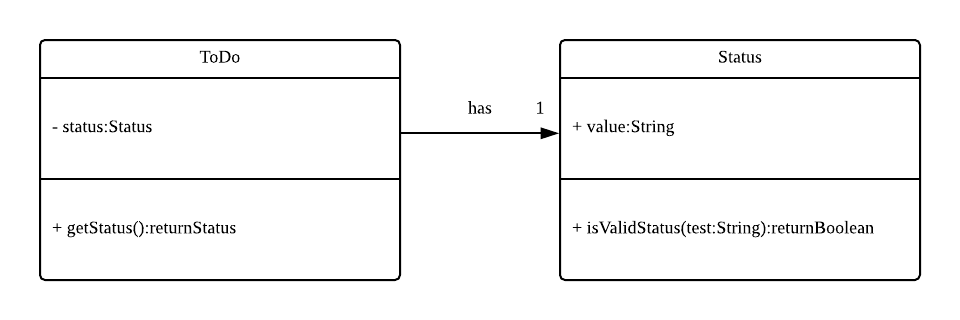


Figure 21. Interaction between ToDo and Status class

When user check/uncheck an existing to-do if specific Index, a new ToDo is created, with the existing ToDo’s Content and appropriate new Status.

The existing to-do is replaced by the new to-do in the UniqueToDoList and the current state of the address book is saved.

### Design Considerations

Aspect: Implementation of AddToDoCommand

* Alternative 1 (current choice): Add a new method setStatus() in ToDo
  + Pros: Fastis does not need to create a new ToDo object to replace the existing to-do.
  + Cons: The implementation does not follow the Single Responsibility Principle.

## Logging

Fastis uses the java.util.logging package for logging. The LogsCenter class is used to manage the logging levels and logging destinations.

Additional information:

* The Logger for a class can be obtained using LogsCenter.getLogger(Class) which will log messages according to the specified logging level
* The logging level can be controlled using the logLevel setting in the configuration file (See [Section 3.5, “Configuration”](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/DeveloperGuide.adoc#Implementation-Configuration))
* Currently log messages are output through both the Console and to a .log file.

Logging Levels

* SEVERE : Critical problems detected which may possibly cause the termination of the application
* WARNING : Non-Critical problems that allows the application to continue, but with caution
* INFO : Information showing the noteworthy actions by the App
* FINE : Details that is not usually noteworthy but may be useful in debugging e.g. print the actual list instead of just its size

## Configuration

Certain properties of the application can be controlled (e.g App name, logging level) through the configuration file (default: config.json).

## NUSmods Integration

### Current Implementation

Fastis uses the available NUSmods API to retrieve module information from the API server.

On startup, the database component makes a connection to the API server and checks the lastmodified field of the JSON file on the server. If the lastmodified date is more recent that the JSON file held in storage, the database component will download and overwrite the existing file on disk.

The Database component then converts the JSON file into a hashMap of modules to be held in the Model

When the schedule of a Person is needed, the Model component will pass the person’s TimeTableLink to the Database component. The Database component will make a URLConnection to the short URL in the TimeTableLink to retrieve the query of the full URL.

QueryParser then takes the query and parses them into modules and lessons, which get passed back to the Model component. DatabaseManager then searches for these modules and lessons and returns a list of Events.

### Design Considerations

Aspect: Storage of Module information

* Alternative 1 (current choice): Store a complete dataset of all modules
  + Pros: Module information will still be available offline
  + Cons: Module information will require additional memory space
* Alternative 2: Store nothing, retrieve only module information of single module each time.
  + Pros: Implementation requires no additional storage and little memory
  + Cons: Implementation will be reliant on good internet connection

## List Group Members Feature

Fastis lists all persons in Fastis that have tags similar to input.

### Current Implementation

Fastis uses ListGroupMembersCommand, which resides under Logic to facilitate the listing of members under the same tag.

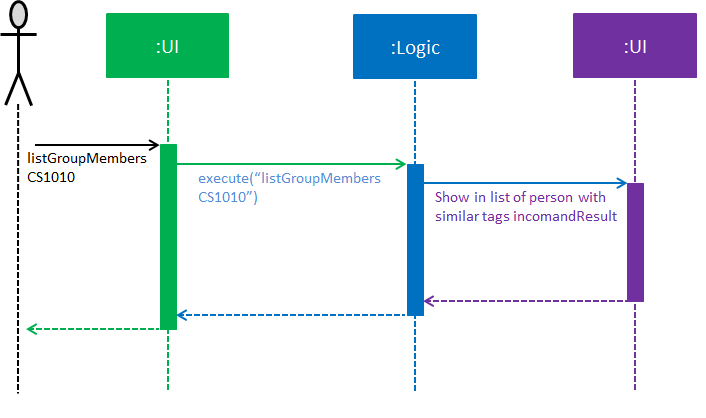


Figure 22. List Group Members sequence diagram

When user types in command line ListGroupMembers or lgm, Fastis will use the keyword provided to search for the group tag and list out all members under the same group tag. The above diagram, Figure 22, shows the sequence of activity.

### Design Considerations

* Alternative 1 (current choice): Use a command to list out the group members.
  + Pros: Easy to change the methods called by command.
  + Cons: Unable to link to group class.
* Alternative 2: Add a new abstract method ListGroupMembersCommand()
  + Pros: Edit ListGroupMembersCommand() easily to suit our needs
  + Cons: Hard for new developers to understand the template pattern.

## Calendar Feature

### Current Implementation

Fastis uses a stand-alone Calendar class, adapted from javafx-calendar by SirGoose3432 (<https://github.com/SirGoose3432/javafx-calendar>).

Two instances of calendar are implemented:

* A monthly calendar to keep track of upcoming events and deadlines.
* A weekly timetable to facilitate scheduling a suitable meet-up.

The current time is retrieved upon startup, and both calendars base on that point in time to display the appropriate time frame.

Both calendars utilize the Event class to determine which slots in the schedule are occupied. Users can associate each event to a Person or a Group, and can modify these events through commands.

The details of these event, e.g. time, location, people involved, are saved locally in .xml file format. It is also retrieved upon start up by the Storage component and saved within the Model for the whole process.

The weekly component of the Calendar is dependent on the NUSMods component for importing the timetable.

### Design Considerations

Aspect: Implementation of the Calendar

* Alternative 1 (current choice): Implement a stand-alone Calendar class
  + Pros:
    - Easy to manipulate data.
    - Easy to customize appearance.
  + Cons:
    - Implementation is likely not optimized.
    - Might not be aesthetically pleasing.
* Alternative 2: Import external libraries/API
  + Choices:
    - [Google Calendar API](https://developers.google.com/calendar/)
    - [CalendarFX](https://github.com/dlemmermann/CalendarFX)
  + Pros:
    - Likely optimized algorithms.
    - Aesthetically pleasing.
  + Cons:
    - Restricted to what the libraries/API offer.
    - Need a firm understanding of the whole API to implement well.

Aspect: Implementation of the Event class

* Alternative 1 (current choice): One Event class for both monthly and weekly calendars.
  + Pros: The codes are concise and can be used interchangeably.
  + Cons: Prone to flawed and buggy implementations where methods are used where they are not supposed to.
* Alternative 2: One Event interface with 2 classes implementing it accordingly.
  + Pros: The codes are modularized and safe from buggy behaviours.
  + Cons: There is likely a lot of code to write.

## Add Group feature

Fastis adds a group named by the user from input.

### Current Implementation

The add groups mechanism is facilitated by AddGroupCommand, which resides inside Logic component. It supports adding Group objects to the address book. AddGroupCommand inherits from UndoableCommand.

Hence, AddGroupCommand can be undone using UndoRedoStack. With the extra layer, the AddGroupCommand that is undoable is implemented this way:

public abstract class UndoableCommand extends Command {

@Override

public CommandResult execute() {

// ... undo logic ...

executeUndoableCommand();

}

}

public class AddGroupCommand extends UndoableCommand {

@Override

public CommandResult executeUndoableCommand() {

// ... delete logic ...

}

}

The groups in the group list are facilitated by Group class. Each Group object have a Information object, representing the information of the group. Address book stores all groups in UniqueGroupList. Group, Information and UniqueGroupListclass reside inside Model component. The following is the class diagram showing the relationship between Group and Information(Figure 23):

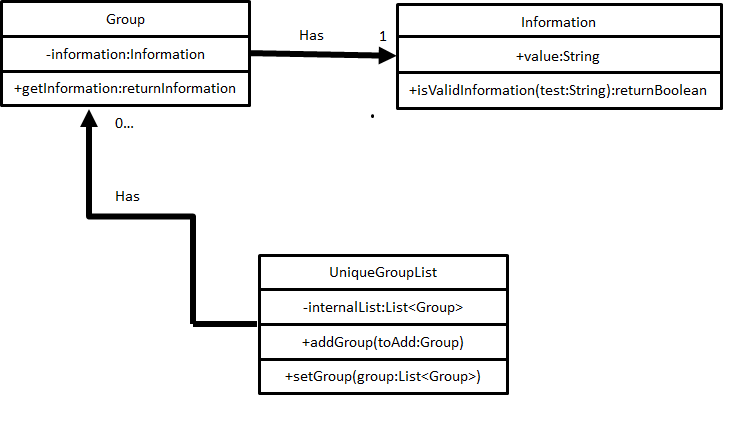


Figure 23. Interaction between group and information

Suppose that the user has just launched the application. The UniqueGroupList in the address book will be empty if no groups have been added before.

The user executes a new AddGroupCommand with Information, to add a new group to the address book. The new group is added to the UniqueGroupList and the current state of the address book is saved. The following sequence diagram shows how the addGroup operation works(Figure 24):

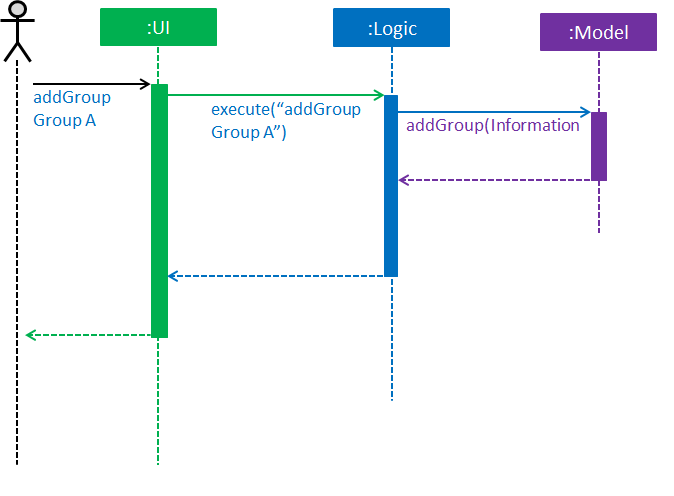


Figure 24. Add group sequence diagram.

### Design Considerations

Aspect: Implementation of AddGroupCommand

* Alternative 1 (current choice): Add a new command method AddGroupCommand()
  + Pros: It is easy for developers to modify method to suit what they want
* Alternative 2: Add a new abstract method executeAddGroupCommand()
  + Pros: It is not able to lose any addGroup functionality as it is now part of the default behavior. Classes that deal with AddGroupCommand do not have to know that executeAddGroupCommand() exist.
  + Cons: It is hard for new developers to understand the template pattern.

# Documentation

This section provides necessary information for developers to edit and publish related documentations, such as User Guide and Developer Guide, for Fastis.

Fastis uses asciidoc for documentation.

|  |  |
| --- | --- |
| ℹ️ | Asciidoc is chosen over Markdown because asciidoc, although a bit more complex than Markdown, provides more flexibility in formatting. |

## Editing Documentation

See [UsingGradle.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingGradle.adoc#rendering-asciidoc-files) to learn how to render .adoc files locally to preview the end result of your edits. Alternatively, you can download the AsciiDoc plugin for IntelliJ, which allows you to preview the changes you have made to your .adoc files in real-time.

## Publishing Documentation

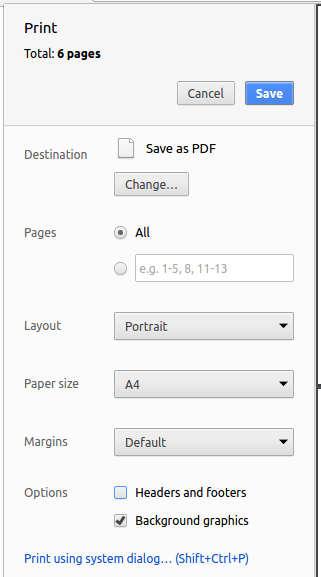
See [UsingTravis.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingTravis.adoc#deploying-github-pages) to learn how to deploy GitHub Pages using Travis.

## Converting Documentation to PDF format

The project uses [Google Chrome](https://www.google.com/chrome/browser/desktop/) for converting documentation to PDF format, as Chrome’s PDF engine preserves hyperlinks used in webpages.

Here are the steps to convert the project documentation files to PDF format:

1. Follow the instructions in [UsingGradle.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingGradle.adoc#rendering-asciidoc-files) to convert the AsciiDoc files in the docs/ directory to HTML format.
2. Go to your generated HTML files in the build/docs folder, right click on them and select Open with → Google Chrome.
3. Within Chrome, click on the Print option in Chrome’s menu.
4. Set the destination to Save as PDF, then click Save to save a copy of the file in PDF format. For best results, use the settings indicated in the screenshot below (Figure 25).

Figure 25. Saving documentation as PDF files in Chrome

# Testing

This section documents how to run the tests on Fastis.

## Running Tests

There are three ways to run tests.

Method 1: Using Gradle in headless mode (recommended)

Thanks to the [TestFX](https://github.com/TestFX/TestFX) library, GUI tests can be run in the headless mode. In the headless mode, GUI tests do not show up on the screen. That means you can do other things on the Computer while the tests are running.

To run tests in headless mode, open a console and run the command gradlew clean headless allTests (Mac/Linux: ./gradlew clean headless allTests)

|  |  |
| --- | --- |
| 💡 | The above-mentioned method is the most reliable. The other two listed below might fail some GUI tests due to platform/resolution-specific idiosyncrasies. |

Method 2: Using Gradle

Using Gradle without the headless mode will cause GUI tests to show up on screen. The test functions will simulate mouse movement and keyboard keystrokes, and tests might fail if you move your mouse or type anything on the keyboard. Thus during the GUI tests it is recommended that you leave your machine alone.

To run the tests, open a console and run the command gradlew clean allTests (Mac/Linux: ./gradlew clean allTests)

|  |  |
| --- | --- |
| ℹ️ | See [UsingGradle.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingGradle.adoc) for more info on how to run tests using Gradle. |

Method 3: Using IntelliJ JUnit test runner

Using this method, GUI tests will also show up on screen, and will require you to stop mouse and keyboard activity in order to run successfully.

This method requires you to have the project open on IntelliJ:

* To run all tests: Right-click on the src/test/java folder and choose Run 'All Tests'
* To run a subset of tests: Right-click on a test package, test class, or a single test and choose Run 'TEST\_NAME'

## Types of tests

Fastis has two types of tests:

* GUI Tests - These are tests involving the GUI. They include:
  + *System Tests* that test the entire App by simulating user actions on the GUI. These are in the systemtests package.
  + *Unit Tests* that test the individual components. These are in seedu.address.ui package.
* Non-GUI Tests - These are tests not involving the GUI. They include:
  + *Unit tests* that target the lowest level methods/classes.

(e.g. seedu.address.commons.StringUtilTest)

* + *Integration tests* that check the integration of multiple code units (those code units are assumed to be working).

(e.g. seedu.address.storage.StorageManagerTest)

* + Hybrids of unit and integration tests. These tests check multiple code units, as well as how the are connected together.

(e.g. seedu.address.logic.LogicManagerTest)

## Troubleshooting Testing

Problem: HelpWindowTest fails with a NullPointerException.

* Reason: One of its dependencies, UserGuide.html in src/main/resources/docs is missing.
* Solution: Executes Gradle task processResources.

# Dev Ops

This section documents tools for building, testing and releasing new versions of Fastis more efficiently.

## Build Automation

See [UsingGradle.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingGradle.adoc) to learn how to use Gradle for build automation.

## Continuous Integration

Fastis uses [Travis CI](https://travis-ci.org/) and [AppVeyor](https://www.appveyor.com/) to perform Continuous Integration on our projects. See [UsingTravis.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingTravis.adoc) and [UsingAppVeyor.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingAppVeyor.adoc) for more details.

## Coverage Reporting

Fastis uses [Coveralls](https://coveralls.io/) to track the code coverage of our projects. See [UsingCoveralls.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingCoveralls.adoc) for more details.

## Documentation Previews

When a pull request has changes to asciidoc files, developers can use [Netlify](https://www.netlify.com/) to see a preview of how the HTML version of those asciidoc files will look like when the pull request is merged. See [UsingNetlify.adoc](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingNetlify.adoc) for more details.

## Making a Release

Here are the steps to create a new release:

1. Update the version number in [MainApp.java](https://github.com/CS2103JAN2018-W15-B3/main/src/main/java/seedu/address/MainApp.java).
2. Generate a JAR file [using Gradle](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/UsingGradle.adoc#creating-the-jar-file).
3. Tag the repo with the version number. e.g. v0.1
4. [Create a new release using GitHub](https://help.github.com/articles/creating-releases/) and upload the JAR file you created.

## Managing Dependencies

A project often depends on third-party libraries. For example, Fastis depends on the [Jackson library](http://wiki.fasterxml.com/JacksonHome) for XML parsing. Managing these dependencies can be automated using Gradle. For example, Gradle can download the dependencies automatically, which is better than these alternatives:

* Including those libraries in the repo (this bloats the repo size)
* Requiring developers to download those libraries manually (this creates extra work for developers)

# Appendix A: Product Scope

This section provides detailed description of Fastis’s target user, value propositions and current features.

Target user profile: NUS students with group projects

* Have a number of information to keep track of:
  + Group members’ contact information
  + Group members’ timetables
  + Projects’ to-dos
  + Meetups time
* Frequently use desktops and laptops
* Type a lot
* Use NUSMods for their timetable
* Have to schedule meetups frequently for group projects
* Have a lot of group members from group

Value proposition: Help students manage group project effectively

Feature contribution

* Detail field for person (minor):
  + Detail for person is additional information such as remark, hobbies, comments, etc
  + Detail allows user to add information that does not fall unser categories such as phone, email, etc
  + Detail allows storing person information to become more flexible
* Timetable link field for person (minor):
  + Timetable link for person is an NUSMods website link
  + Timetable link displays the actual link of NUSMods website shown when person is selected
  + Timetable link allows user to manage NUSMods link of person
* Customizable tags color (minor):
  + User can set color of specific tags
  + Customizable tags color allows user to better personalise the application
* Dark color theme for application bar and background (minor):
  + Application bar and background are changed to dark grey color
  + Dark color theme makes the application comfortable to use in different light conditions
* Personal To-do list (major):
  + User can add to-dos, notes, upcomming tasks and organize them in the todo-list
  + To-do list serves to remind user of important tasks, events in group projects
* Meetup Time Generator (major):
  + Generator parses information from persons' NUSMods timetables and generate a suitable project meetup time
  + Meetup Time Generator allows user to quickly decide meetup time without manually checcking timetables
* Meetup Calendar (major):
  + User can add meetups with specific start, end time and display meetups on the calendar
  + Meetup Calendar allows user to visualise and remember upcomming group events/meetups
* Organize persons in groups (major):
  + User can store specific persons in project groups
  + User can have a group list and display persons in each group
  + Groups allows user to manage persons based on the project groups they belongs to

# Appendix B: User Stories

This section documents the user stories that were brainstormed for Fastis’s target users.

Priorities: High (must have) - \* \* \*, Medium (nice to have) - \* \*, Low (unlikely to have) - \*

|  |  |  |  |
| --- | --- | --- | --- |
| Priority | As a …​ | I want to …​ | So that I can…​ |
| \* \* \* | Student with group project | Add person with project group tag | I know which person belongs to which project groups |
| \* \* \* | Student that is finished with a group project | Delete the group tag of a finished group project | I will not see the group in the application anymore |
| \* \* \* | Student who forget teammates’ information | Find a person by name | I can find out more details of the person such as location, contact number |
| \* \* \* | Student who made a mistake in recording a group tag | Edit a person’s group tag | I can change the group tag accordingly |
| \* \* \* | Student who recorded the wrong personal information | Edit a person | I can correct the details of the person |
| \* \* \* | Student who wants to know which teammates are in the project group | Print out the list of teammates under the same group tag | I know which teammates I am meeting |
| \* \* \* | Student with arranged meet-ups | Add meet-ups with title, time, venue to the calendar | Have the summary of upcoming meet-ups in a quick glance |
| \* \* \* | Student using CLI | Press up button to copy the previous command | I do not need to retype duplicate commands |
| \* \* \* | Student that is involved with multiple groups | search events by its title | I can get details of a particular meet-up |
| \* \* \* | Student who wants to arrange meet-ups | Search meet-ups by its title | I can get details such as time and place of a particular meet-up |
| \* \* | Student who wants to know the location of teammates | Find the location of teammates’ address via google maps | Decide on a appropriate meeting location for all teammates considering their home address |
| \* \* \* | Student adding teammates’ information | Add the link to teammates’ timetable | I can see teammates’ timetables to arrange meet-ups |
| \* \* \* | Student working with new teammates | Add their contact information | I can contact them if the need arises |
| \* \* \* | user | add info of the members involved in events | So that I know who I would need to talk to |
| \* \* \* | Student first time using the application | See the usage instructions | Learn how to use the application |
| \* \* | Student that does not leave applications open | See an overview of the week’s meetup right away when the application opens | Do not have to type in any commands when I first open the application |
| \* \* \* | Student who have unused contacts | Delete a teammate from the application | I can free up storage for my application |
| \* \* \* | Busy student with busy schedule | Have a reminder of the upcoming project meeting | I can be reminded of impending project with the details of group members printed on it |
| \* \* | Student who would like different colours tag for different projects | Have customizable coloured tags for different groups | Easily differentiate the groups via colour tags |
| \* | Student who is too lazy to eyeball through the timetables | Have an appropriate meet-up time generated for a group | I do not have to manually come up with time for meetup |
| \* \* \* | Student who created a group | Show teammates from a group | I can see the information of the teammates from the group |
| \* \* \* | Student who has multiple group projects | Retrieve a list of all my groups | I can see all my groups at once |
| \* \* \* | Student who is finished with a project | Delete everyone in a group in one go | I do not have to delete contacts one by one |
| \* \* \* | Student who is too lazy to type | Use a shorter version of a command | Use the app faster |
| \* | Student who uses NUSMODS | Use my nusmods link to add my schedule into Fastis | I don’t have to manually input my timetable |
| \* \* | User with accessibility problems | Increase the font size of the application | I can read and see more easily |
| \* | Students who likes other colours on the interface | Change color scheme | Personalise the app |
| \* | Student who wants to call an absent teammate | Open teammates’ WHATSAPP page within the application e | I can alert teammates of the meeting |
| \* | Student who do not how to reach the destination of the meet-up | Have the venue of the meet-up shown on google map | I can know the direction to the meetup |
| \* | user | find the road that travels the shortest distance to the location of event | So that I can rely on the, map while driving |
| \* | user | link an event with another | So that events that are related can be linked together so that I know which events are related |
| \* | user | play music from address book | So that I can listen to music I want |
| \* | user | play videos from YouTube | So that I can watch videos I want |

# Appendix C: Use Cases

This section documents the use cases of Fastis’s current features. For all use cases below, the System is the Fastis and the Actor is the user, unless specified otherwise.

**Use case: Add Meet up**

MSS

1. User requests to add a meet up by entering information
2. Fastis adds the meet up to the calendar
3. Use case ends.

Extensions

* 2a. Meet up clashes with an existing event.
  + 2a1. Fastis notifies user and asks for confirmation.
  + 2a2. User enters confirmation.

Use case resumes from 2.

**Use case: Delete Meet up**

MSS

1. User requests to lists all meet ups.
2. Fastis shows all event, listed with indexes.
3. User requests to delete a specific meet up in the list.
4. Fastis deletes the meet up.

Use case ends.

Extensions

* 2a. Calendar is empty.
  + 2a1. Fastis notifies user that the calendar is empty.

Use case ends.

* 3a. Index given is invalid.
  + 3a1. Fastis notifies user that the index was invalid.

Use case resumes from 3.

**Use case: Group multiple persons into a group**

MSS

1. User requests to list all persons.
2. Fastis shows all persons, listed with an index.
3. User requests to add a group tag to a few specified persons on the list.
4. Fastis adds the new group to the group list.

Use case ends.

Extensions

* 2.a Index given is invalid.
  + 2a1. Fastis notifies user that the index given was invalid.

Use case resumes from 1.

**Use case: List all persons/groups**

MSS

1. User requests to list all persons or all groups in Fastis.
2. Fastis lists all persons or all groups.

Use case ends.

Extensions

* 2a. The persons list is empty.
  + 2a1. Fastis notifies user that the persons list is empty.

Use case ends.

* 2b. No group tags were found.
  + 2b1. Fastis notifies user that no groups were found.

Use case ends.

**Use case: Delete all persons in a group**

MSS

1. User requests list all groups.
2. Fastis shows a list of all groups.
3. User requests to delete all persons in a specific group in the list
4. Fastis deletes all persons in the group, and deletes the group tag

Use case ends.

Extensions

* 2a. The list is empty.
  + 2a1. Fastis notifies user that no groups were found.

Use case ends.

* 3a. The given index is invalid.
  + 3a1. Fastis shows an error message.

Use case resumes from 2.

**Use case: Show a person’s timetable**

MSS

1. User requests to show the timetable of the specified person.
2. Fastis show the timetable for the person.

Use case ends.

Extensions

* 2a. The specified person has no timetable
  + 2a1. Fastis notifies the user that the person has no timetable

Use case ends.

**Use case: Print out all teammates under the same group**

MSS

1. User requests to list the teammates for a particular group
2. Fastis shows the table with group heading and lists all members in it.

Use case ends.

Extensions

* 2a. There is no such group in Fastis.
  + 2a1. Fastis notifies user that no group was found.

Use case ends.

**Use case: Help**

MSS

1. User requests to see all commands
2. Fastis shows all commands, sorted by name
3. User types in which command they want to view
4. Fastis shows the manual of that command

Use case ends.

Extensions

* 2a. Command name given is invalid.
  + 2a1. Fastis notifies user that the command was invalid.

Use case resumes from 3.

**Use case: Copy Previous Command**

MSS

1. User enters keystroke to navigate to previously entered command
2. Fastis copies the previous command into the input field

Use case ends.

**Use case: Change Tag Color**

MSS

1. User enter command to change a specific tag color
2. Fastis changes the color accordingly and display it

Use case ends.

Extensions

* 2a. The tag specified doesn’t exist within the address book
  + 2a1. Fastis notifies the user.

Use case ends

* 3a. The color specified is not supported by Fastis
  + 3a1. Fastis notifies the user.

Use case ends

*{More to be added}*

# Appendix D: Non Functional Requirements

This section documents Fastis’s non-functional requirements, which are additional criterias for Fastis to serve the target users well.

* Fastis should work on any [mainstream OS](https://github.com/CS2103JAN2018-W15-B3/main/blob/master/docs/DeveloperGuide.adoc#mainstream-os) as long as it has Java 1.8.0\_60 or higher installed.
* Fastis should be able to hold up to 1000 persons without a noticeable sluggishness in performance for typical usage.
* A user with above average typing speed for regular English text (i.e. not code, not system admin commands) should be able to accomplish most of the tasks faster using commands than using the mouse.
* Fastis should usable without the need of a mouse.
* Fastis should be usable solely via a command line interface.
* Fastis should respond within 2 seconds.
* Fastis should have an easy to follow user guide.
* Fastis should open the help page when user enters an invalid entry.
* Fastis should be possible to fixed and debugged in the event of malfunction.

*{More to be added}*

# Appendix E: Glossary

This section defines terms and definitions that are frequently used in this Developer Guide.

*Mainstream OS*

Windows, Linux, Unix, OS-X

*Private contact detail*

A contact detail that is not meant to be shared with others

*Event*

A set of scheduled activity that user needs to attend at a specific time E.g.: Interviews, parties, CCAs, talks, coding challenges, assignments, etc.

# Appendix F: Instructions for Manual Testing

This section provides instructions to test the app manually.

|  |  |
| --- | --- |
|  | These instructions only provide a starting point for testers to work on; testers are expected to do more exploratorytesting. |

## F.1. Launch and Shutdown

1. Initial launch
   1. Download the jar file and copy into an empty folder
   2. Double-click the jar file

Expected: Shows the GUI with a set of sample contacts. The window size may not be optimum.

1. Saving window preferences
   1. Resize the window to an optimum size. Move the window to a different location. Close the window.
   2. Re-launch the app by double-clicking the jar file.

Expected: The most recent window size and location is retained.

*{ more test cases …​ }*

## F.2. Deleting a person

1. Deleting a person while all persons are listed
   1. Prerequisites: List all persons using the list command. Multiple persons in the list.
   2. Test case: delete 1

Expected: First contact is deleted from the list. Details of the deleted contact shown in the status message. Timestamp in the status bar is updated.

* 1. Test case: delete 0

Expected: No person is deleted. Error details shown in the status message. Status bar remains the same.

* 1. Other incorrect delete commands to try: delete, delete x (where x is larger than the list size) {give more}

Expected: Similar to previous.

*{ more test cases …​ }*

## F.3. Saving data

1. Dealing with missing/corrupted data files
   1. *{explain how to simulate a missing/corrupted file and the expected behavior}*

*{ more test cases …​ }*