SpinBox Project Portfolio

Andrew Lau Jia Jun (A0182815B)

About the Project

SpinBox is a school life management desktop application which I have created alongside 3 other year 2 computer engineering students. It is an application that is morphed from a basic command line interface desktop task management application. SpinBox (Figure 1) is aimed at NUS students who are seeking assistance in managing school life – from handling deadlines and keeping track of deliverables and important dates, to monitoring module updates and viewing grade breakdowns for every module.

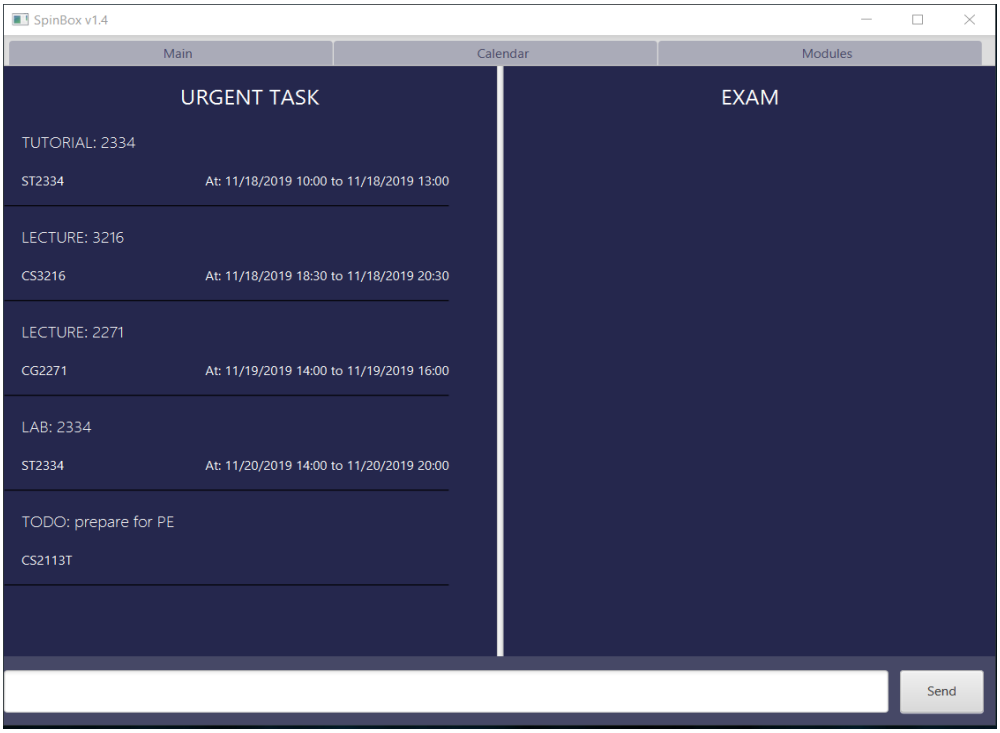

Figure 1. UI of SpinBox

My role was to implement support for an intuitive command parsing and execution approach in order to ease the user experience. On top of that, I was also tasked with the implementation of the set and multiple features. The purpose of this document is to document my contribution to the project. The following sections will elucidate on these enhancements in greater detail through illustrations, as well as the documentation on these features, which I have included to the user and developer guides.

Note the following symbols and formatting used in this document:

1. command : A grey highlight (also known as mark-up) indicates a command that can be typed into the command line as an input and be executed by SpinBox.
2. Text : Blue text with grey highlight indicates a component, class or object in the architecture of SpinBox

Summary of Contributions

This section contains a summary of my code, documentation and other contributions to the team project.

Enhancements Added

1. Intuitive Command Interpreter (Pull Request #76, #83, #191, #214)

- What it does: Instead of having to type the full command for every command wants to execute, the user (NUS students) can provide a shorter input command and SpinBox will be able to perform a context switch to the page which the user is currently at, and execute the command.
- Justification: As there are multiple pages and tabs in SpinBox, users may find it inconvenient and strenuous if they are required to provide the full command all the time, especially if the input is lengthy. Hence, this enhancement serves to provides a form of convenience to the users as they are now only required to provide certain keywords as the input command.
- Highlights: This enhancement weaves in well with existing commands such as “add” and “remove”, as well as other novel features developed throughout the course of this project, such as “set-name” and “view”. The implementation was challenging as other alternatives were considered and attempted before the final current implementation.

2. Enhanced Versatility of Module Components Management (Pull Request #137, #230)

- What it does: It allows the user to be able to edit the date and time details of a schedulable type task.
- Justification: As there is a likelihood of the user facing the situation of having to reschedule a task, this feature allows the user to enjoy a more versatile configuration of the application, without having to remove the original component and recreate it whenever certain details of the component are changed.
- Highlights: This enhancement enriches the user experience with SpinBox, as it increases the convenience of using this application and displays flexibility in allowing the user to modify the existent components.

Code contributed

Please click on these links to see a sample of my code: [Project Code Dashboard]

Other Contributions

- **Community**
  - Reviewed Pull Requests with Non-Trivial Comments (Pull Requests #70, #96)
  - Fixed bugs that are present in the application (Pull Requests #103, #124, #176)
- **Documentation**
  - Created the AboutUs and README page (Pull Requests #22, #32)
  - Designed the Ui Mock-up images for the modules page (Pull Request #65, #143)

- **Enhancement to Existing Features**
    - Created the help page with a comprehensive explanation to aid user in the utilization of the application (Pull Request #116)
    - Implemented the different schedulable type tasks such as Exam, Lab, Tutorial etc and the multiple update and removal features. (Pull Requests #64, #69, #108, #124, #128)
    - Implemented the command to allow the removal of a module, as well as its stored components (#194)
    - Implemented the set-name command to allow users to edit the description of a task/file/note. (#130)
    - Refactored parts of the code (Pull Requests #124, #128, #214, #230)
    - Wrote additional tests for existing features to increase coverage by 25% (Pull requests #183, #185, #191, #217)
- **Project Management**
    - Set the milestone dates on Github to better keep track of the project deadlines
    - Took charge of the documentation aspects of the team project

Contributions to the User Guide

As our team decided to venture the path of morphing the original application into SpinBox, I proceeded to update the initial User Guide with the instructions for the enhancement features implemented. The following excerpts reflect parts of my contribution to the SpinBox User Guide, namely the symbols explanation, command format, `set-name`, `set-date` as well as the command summary (Figure 2).

3. Features

Note the following symbols and formatting used in this document:

1. ℹ️: Used to provide information regarding certain errors that you may face while using SpinBox.
2. 💡: Used to denote an important tip that further enhances your user experience.

Command Format
• Words in `[UPPER_CASE]` are the parameters to be supplied by the user e.g. in `view / modules CG1111 [TYPE]`, `[TYPE]` is a parameter which can be used as `view / modules CG1111 tasks`. • `[INDEX]` refers to the index number shown in the displayed file/grade/note/task list. The index **must be a positive integer**, e.g. 1,2,3, … • `[DATETIME]` is to be entered as `MM/DD/YYYY HH:MM`, and can be substituted using natural language, e.g. `tomorrow 6pm`. **However, the accuracy of natural language input cannot be guaranteed**. We strongly recommend sticking to our specified format for best results.

💡 Pressing the ↑ and ↓ arrows will display the previous and next input respectively in the command box.

3.5.1.12. Editing the name of a task : set-name / task

Edit the name of a task under the selected module to a new name. Alternatively, use the full version if the module is not currently selected.

Format: `set-name / task [INDEX] to: [NAME]` or `set-name [MODULE_CODE] / task [INDEX] to: [NAME]`

- Edits the name of the task at the specified `[INDEX]`.
- The `[INDEX]` can be seen from the command `view / tasks`

Example:

- `set-name / task 1 to: return book` (Name of first task under current module edited to return book)
- `set-name CG1111 / task 2 to: return book` (Name of second task under module CG1111 edited to return book)

💡This command allows you to edit the description of a task, so that you do not have to manually remove a task and add a new task with the updated description.

3.5.1.13. Editing the date of a schedulable task : `set-date / task`

Edit the date of a schedulable task under the selected module to a new date. Alternatively, use the full version if the module is not currently selected.

Format for single datetime tasks: `set-date / task [INDEX] to: [DATETIME]` or `Set-date [MODULE_CODE] / task [INDEX] to: [DATETIME]`

Format for double datetime tasks: `set-date / task [INDEX] to: [DATETIME to DATETIME]` or `set-date [MODULE_CODE] / task [INDEX] to: [DATETIME to DATETIME]`

- Edit the date of the schedulable task at the specified `[INDEX]`.
- The `[INDEX]` can be seen from the GUI, next to each task.

Example:

- `set-date / task 1 to: 01/01/2019 01:00 to 02/01/2019 01:00` (First task under currently selected module set to new datetime)
- `Set-date CG1111 / task 2 to: 01/01/2019 01:00 to 02/01/2019 01:00` (Second task under module CG1111 set to new datetimes)

ℹ️Attempting to edit the date of a non-schedulable task (i.e. To-do) will result in an error message.

5. Command Summary

Category	Command	Command Format and Example
General	bye	Format: `bye` Example: `bye`
	view	Format: `view / [PAGE]` Example: 1. `view / main` 2. `view / calendar`
	help	Format: 1. `help` 2. `help / [COMMAND]` Example: 1. `help` 2. `help / set-name`
	find	Format: 1. `find [PAGE_CONTENT] / [ITEM_TYPE] [KEYWORD]` Example: 1. `find CG1112 / file tutorial` 2. `find / task exam`
	export	Format: 1. `export [ITEM_TYPE]` 2. `export [MODULE_CODE] / [ITEM_TYPE]` Example: 1. `export / files` 2. `export CG1111 / grades`
	populate	Format: `populate` Example: `populate`
	cap [coming in v2.0]	Format: `bye` Example: `bye`
Module Section	view	Format: 1. `view / modules` 2. `view / [MODULE_CODE]` 3. `view / modules [MODULE_CODE]` 4. `view / tasks` 5. `view / [MODULE_CODE] tasks` 6. `view / files` 7. `view / [MODULE_CODE] files` 8. `view / grades` 9. `view / [MODULE_CODE] grades` Example: 1. `view / modules` 2. `view / CG1111` 3. `view / modules CG1111` 4. `view / tasks` 5. `view / CG1111 tasks` 6. `view / files` 7. `view / CG1111 files` 8. `view / grades` 9. `view / CG1111 grades`
	add	Format: Adding a module: 1. `add / module [MODULE_CODE] [MODULE_DESCRIPTION]` Adding a task/file/grade/note (include `[MODULE_CODE]` after `add` if the module is not currently selected): 2. `add / todo [DESCRIPTION]` 3. `add / deadline [DESCRIPTION] by: [DATETIME]` 4. `add / [TASKTYPE] [DESCRIPTION] at: [DATETIME]` * Possible `[TASKTYPE]`: Event/Exam/Lab/Lecture/Tutorial 5. `add / file [FILENAME]` 6. `add / grade [COMPONENT] weightage: [WEIGHTAGE]` 7. `add / note [MESSAGE]` Example: 1. `add / module CG1111 EPP1` 2. `add / todo Return book` 3. `add CG1111 / todo Return book` 4. `add / deadline Submit assignment by: 10/04/2019 23:59` 5. `add / event Project Showcase at: 10/05/2019 10:00 to 10/05/2019 12:00` 6. `add / file file1` 7. `add / grade Written Report weightage 12.5` 8. `add / note Send email`

Figure 2. Snippet of the Command Summary

Contributions to the Developer Guide

The following section details on my contributions to the SpinBox Developer Guide, more specifically for the implementation of the intuitive command interpreter parser.

3.1.1 Implementation Process

Instead of having to type the full command for every command that the user wants to execute, the user can provide a shorter input command and SpinBox will be able to perform a context switch to the page which the user is currently at, and execute the command.

The implementation of the context-aware command interpreter is carried out in the Parser, since every command that the user inputs will first be passed into Parser. The user input will then be passed into parse for processing into a full workable command.

Given below is an example usage scenario and how the context-aware command interpreter feature behaves at each step.

Step 1. The user launches SpinBox and executes view / modules CG1111, assuming that it is not the first time that the user is launching SpinBox, and there is already an existent module in the storage with the module code “CG1111”.

Step 2. The user executes add modules CG1111 / todo Prepare for class, with the intention of adding a new to-do task under the module CG1111. Since the format of a standard command is <action> <page data> / <content>, the user input command will first be split by "/" into the String array slashSeparate (Figure 6).

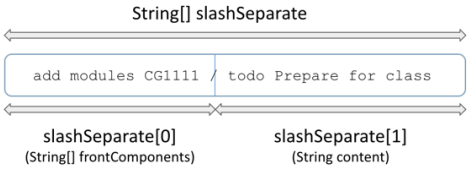

Figure 6. String array slashSeparate

Step 3. Since the length of the String array slashSeparate is larger than one (length is two) and the first word of the input is not "help", all the input which comes after the "/" (slashSeparate[1]) is stored in the String content. The String array frontComponents will store the input before the "/", which is further split by " ". The String action will store the first array element in frontComponents, while the String pageData will store the input that comes after the action, but before the slash "/" (Figure 7).

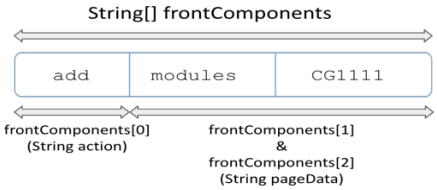

Figure 7. String Array frontComponents

Step 4. pageData is now trimmed and set to lowercase before it is passed into the method commandBuilder for further processing. The reason why it is set to lowercase is to allow the user to have the flexibility in terms of the formatting style of the input. Given that the input is valid and no exception is thrown, commandBuilder is used to build the full page data information for the purpose of allowing the system to be aware of the current context in which the user is in. This is a critical part of the feature, as it contains the crux of allowing the SpinBox to be context-aware. The flow of what transpires in commandBuilder is illustrated in the following activity diagram (Figure 8).

Step 5. commandBuilder first takes in the parameter String inputPageData, which is pageData in step 4. It then proceeds to build the full command based on what is available in the inputPageData, as detailed above. In this case, since the command provided by the user is considered as a full command as it contains both the page details and the module code, commandBuilder will execute the method fullPageComponentAppender that adds "modules" and pageComponent[0] in uppercase (the module code) to pageData before returning the full pageData.

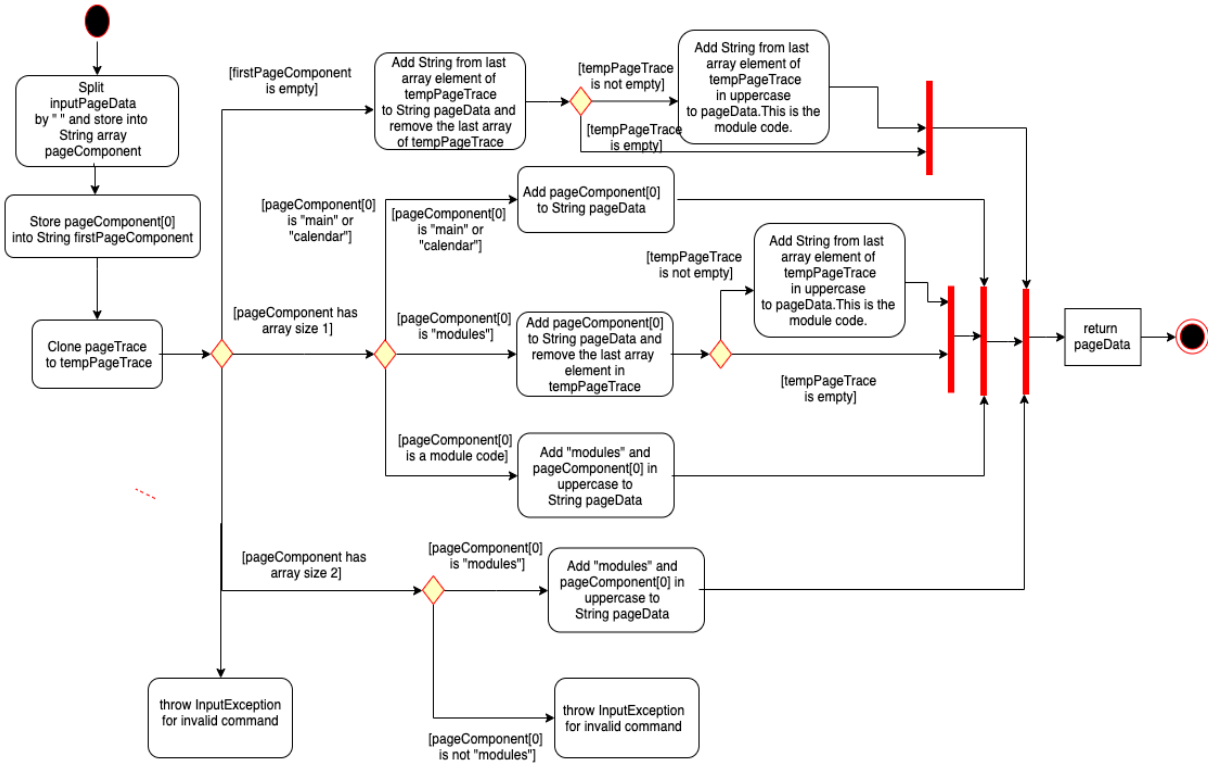

Figure 8. Activity diagram for the method commandBuilder

Step 6. Once commandBuilder returns the full pageData with the relevant current context of the system, pageData will be split by space " " and be stored in the String array pageDataComponents (Figure 9)

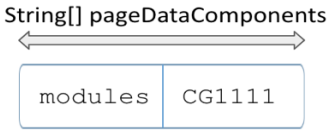

Figure 9. String array pageDataComponents

Step 6. With the action stored in action previously in step 3, a switch statement with action as the parameter will be executed. In this case, since the action is "add", the AddCommand class is invoked and pageDataComponents and content will both be passed in as the parameter. The Command type command will then store the return value, and parse will finally return command**.** The sequence diagram for parse is as illustrated below.

Step 7. Finally, the class SpinBox will print out the return input stored in the form of Command. The String response will then store the String that is returned when the execute() method is invoked in command. The Ui type userInterface then prints Response to the screen and the todo task "prepare for class" is now officially added into the task list under module CG1111.

The following sequence diagram provides a summarisation of the general flow of events when a user provides an input command, and how the context-aware command interpreter dissects and reassembles this command together to form the full command (Figure 10).

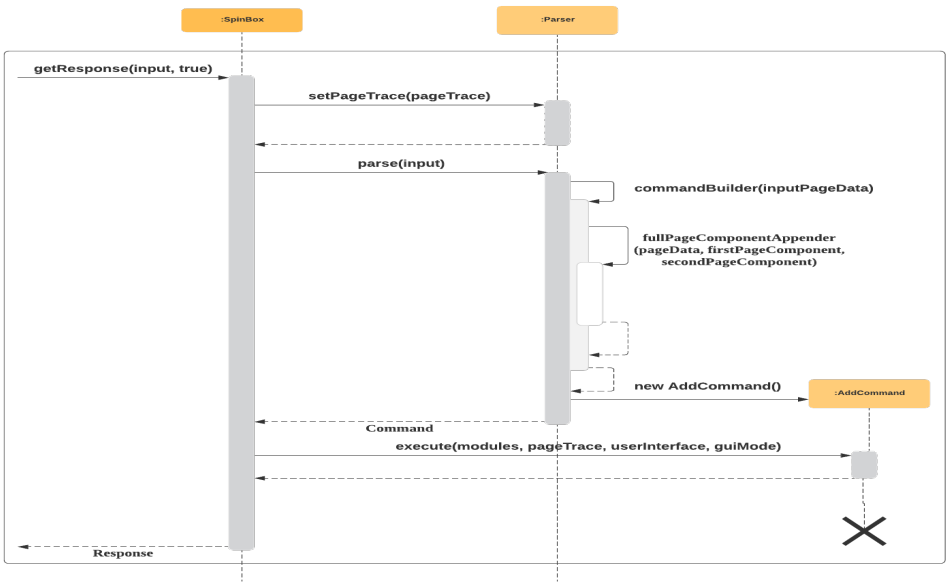

Figure 10. Sequence diagram of the above example

3.1.2 Design Considerations

Aspect	Alternative 1 (Chosen)	Alternative 2
Data Structure of pageTrace (stores the current context of the program)	Using ArrayDeque to represent pageTrace • Pros: An ArrayDeque allows for the access of both the first and the last element, since it is double-ended. • Cons: An ArrayDeque cannot contain a null element in it. It is also not synchronised and hence it does not support concurrent access by multiple threads.	Using Stack to represent pageTrace • Pros: Memory can be dynamically allocated, hence it is easier to add a new element or remove an element from the stack. • Cons: Stack is less accessible as elements are pushed and popped from the top, hence there is no access to the elements underneath
The command format allowed from users	The page details can right after the action word and before the delimiter slash "/", or after the slash and before the content of the command. E.g. view modules CG1111 / tasks VS view / modules CG1111 tasks • Pros: The user can enjoy a highly flexible input format requirement as there are multiple ways of providing an input that leads to the same outcome. • Cons: It may get rather confusing for users due to the multiple possibilities of input format.	For the adding of an item/task, the first word will be the item/task type to be added instead of the action word "add". E.g. todo modules cs2113t task return book • Pros: It is a direct form of output format for the user as there is no need to provide any delimiter. • Cons: The command can get long and complicated, and there will be even more variations of the command to consider. This may potentially cause a misinterpretation of the command by SpinBox and therefore lead to the display of an undesirable output.