Developer Guide

Updated as of 11/11/2019

Prepared by:
CS2113T-F14-4 / CS2101-SC07-GROUP2

1. Introduction

Welcome to the *SpinBox Developer Guide*! In this guide, you can find step by step instructions on setting up SpinBox, as well as ways to modify SpinBox to better suit your needs. You can discover the technical aspects of SpinBox such as its design rationales and implementation details in this document. Interested? Proceed to the next section, Setting Up to get started. We wish you a fruitful experience with SpinBox!

1.1 What is SpinBox?

SpinBox is a desktop companion app for NUS students to provide important, self-contained features that current learning platforms fail to provide. Managing a multitude of tasks, deadlines, on top of a busy schedule can be a strenuous and frustrating experience for many students, especially as the semester progresses into the later weeks. Additionally, needing to frequently context switch between 5-6 modules often takes a toll on a student's ability to perform to the best of his/her abilities.

With SpinBox, we aim to provide our users with the high level view while still being able to dive right into the details. Through our intuitive and powerful Graphical User Interface (GUI), our users can easily manage their tasks, files and grades, classified according to modules. This allows them to focus on what's most important while shielding them from being overwhelmed by context switches and other distractions.

Additionally, they are also able to view your five most urgent tasks across all modules on the main tab, as well as your upcoming exam dates. On the Calendar tab, they will be able to view their tasks for the month, arranged in a chronological fashion.

SpinBox is optimised for students who prefer to work with a Command Line Interface (CLI) while still having the benefits of a GUI. If the user can type fast, SpinBox can organise their tasks, grades and files significantly faster than traditional GUI apps. Through SpinBox, we also hope that our users can hone their prioritization and time management skills, which are beneficial and transferable to the workplace.

SpinBox currently supports Windows, Linux and macOS users.

1.2 Who is this guide for?

You are reading the right document if you are a developer who is interested to learn about the design principles and implementation details behind SpinBox. This guide also provides information that will aid you in better understanding the inner structure and

system of SpinBox, as well as the various classes provided which you can utilise when modifying SpinBox.

2. Setting Up

Follow the step by step instructions below to set up SpinBox project with IntelliJ Integrated Development Environment (IDE).

2.1. Prerequisites

1. JDK 11
2. IntelliJ IDE

ℹ	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.

2.2. Setting up the project on your computer

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
    a. Click `Configure` > `Project Defaults` > `Project Structure`
    b. Click `New…` and find the directory of the JDK
4. Click `Import Project`
5. Locate the `build.gradle` file and select it. Click `OK`
6. Click `Open as Project`
7. Click `OK` to accept the default settings
8. Open a console and run the command `gradlew processResources` (Mac/Linux: `./gradlew processResources`). It should finish with the `BUILD SUCCESSFUL` message. This will generate all resources required by the application and tests.

2.3 Verifying the setup

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

2.4. Configurations to do before writing code

2.4.1. Updating documentation to match your fork

After forking the repo, the documentation will still refer to the AY1920S1-CS2113T-F14-4/main repo.

If you plan to develop this fork as a separate product (i.e. instead of contributing to AY1920S1-CS2113T-F14-4/main), you should do the following:

1. Configure the site-wide documentation settings in build.gradle, such as the site-name, to suit your own project.

2. Replace the URL in the attribute repoURL in DeveloperGuide.adoc and UserGuide.adoc with the URL of your fork.

2.4.2. Setting up CI

Set up Travis to perform Continuous Integration (CI) for your fork. See 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).

ℹ	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).

ℹ	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)

2.5. Getting started with coding

When you are ready to start coding, we recommend that you get some sense of the overall design by reading about SpinBox’s architecture.

3. Design

3.1 Architecture

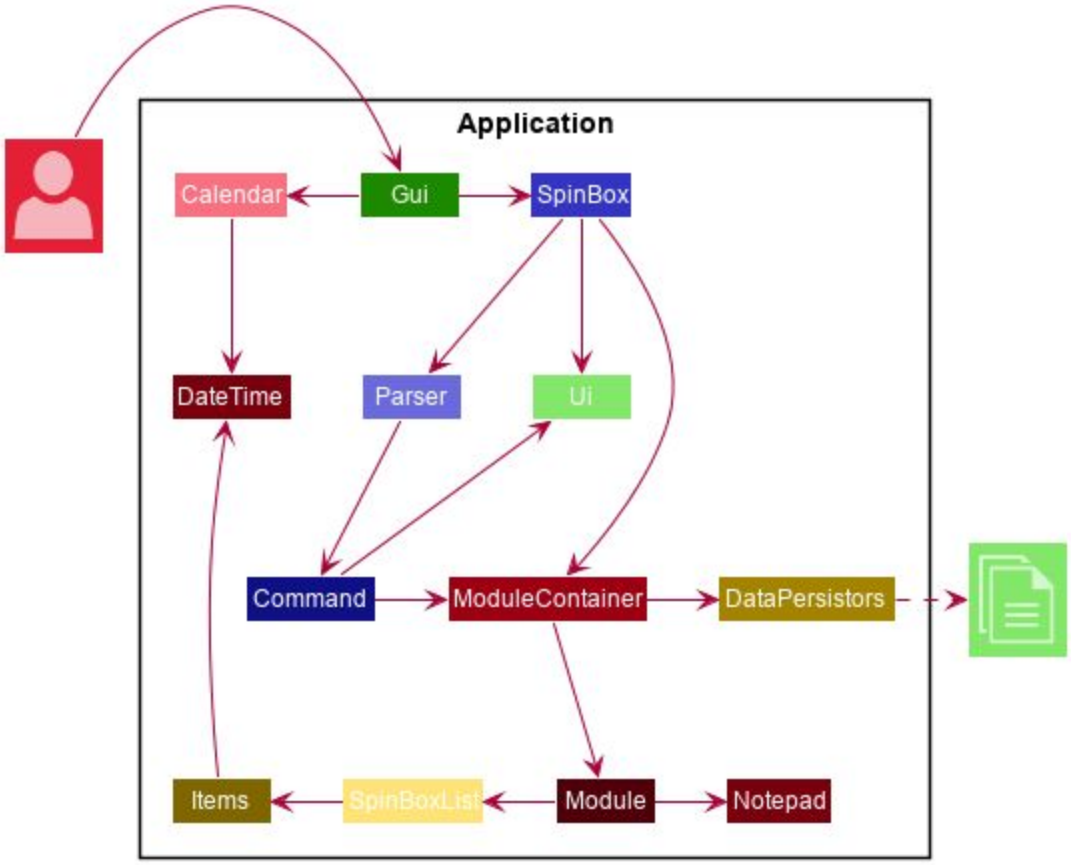

Figure 1.1 Architecture Diagram

This is the ***Architecture Diagram (Figure 1.1)***. It gives the high-level design of the App. Given below is a quick overview of each component.

`Gui` has a `MainWindow` Class which would call `SpinBox`. The `MainWindow` Class is responsible for calling all other `Gui` classes and `Calendar`, for the logic for `Gui` Calendar.

`SpinBox` would call the `Parser` Class to parse input from user. From the input, the `Parser` would be able to return a specific `Command` to `SpinBox`.

It would also interact with the `ModuleContainer` Class and `Ui` Class and pass both of them as parameters into the `Command` Class. This would allow the `Command` class to call functions contained in these two Classes. `ModuleContainers` contains a hashmap of `Module` and also interact with `DataPersistors` and stores the data outside the program.

`Module` contains three `SpinBoxList`, namely `TaskList`, `FileList` and `GradeList`. In addition, each `SpinBoxList` contains `Items` which can be either `GradedComponent` or `File` or a `Task` depending on the type of `SpinBoxList`. If a `Task` is also a `Schedulable`, it would contain two `DateTime` objects for the start and end date and time of the `Schedulable Task`.

`Calendar`, which is one of the deals with logic for `Gui` Calendar, also contain two `DateTime` objects for the start and end date and time of the month it is displaying.

3.1.1 How the architecture components interact with each other

The ***Sequence Diagram (Figure 1.2)*** below shows the sequence diagram for when a user inputs their command into the `Gui`. It shows how SpinBox is able to `setPageTrace()` using `Parser`, before using it to create a `AddCommand` object. The `AddCommand` would be return to `SpinBox`, which would call execute inside `AddCommand` and also pass `ModuleContainer`, and `Ui` objects into the function.

This would then return a response to `SpinBox` which `SpinBox` would be able to update accordingly. The `MainWindow` would also handle the response by updating itself and all `Gui` components to it.

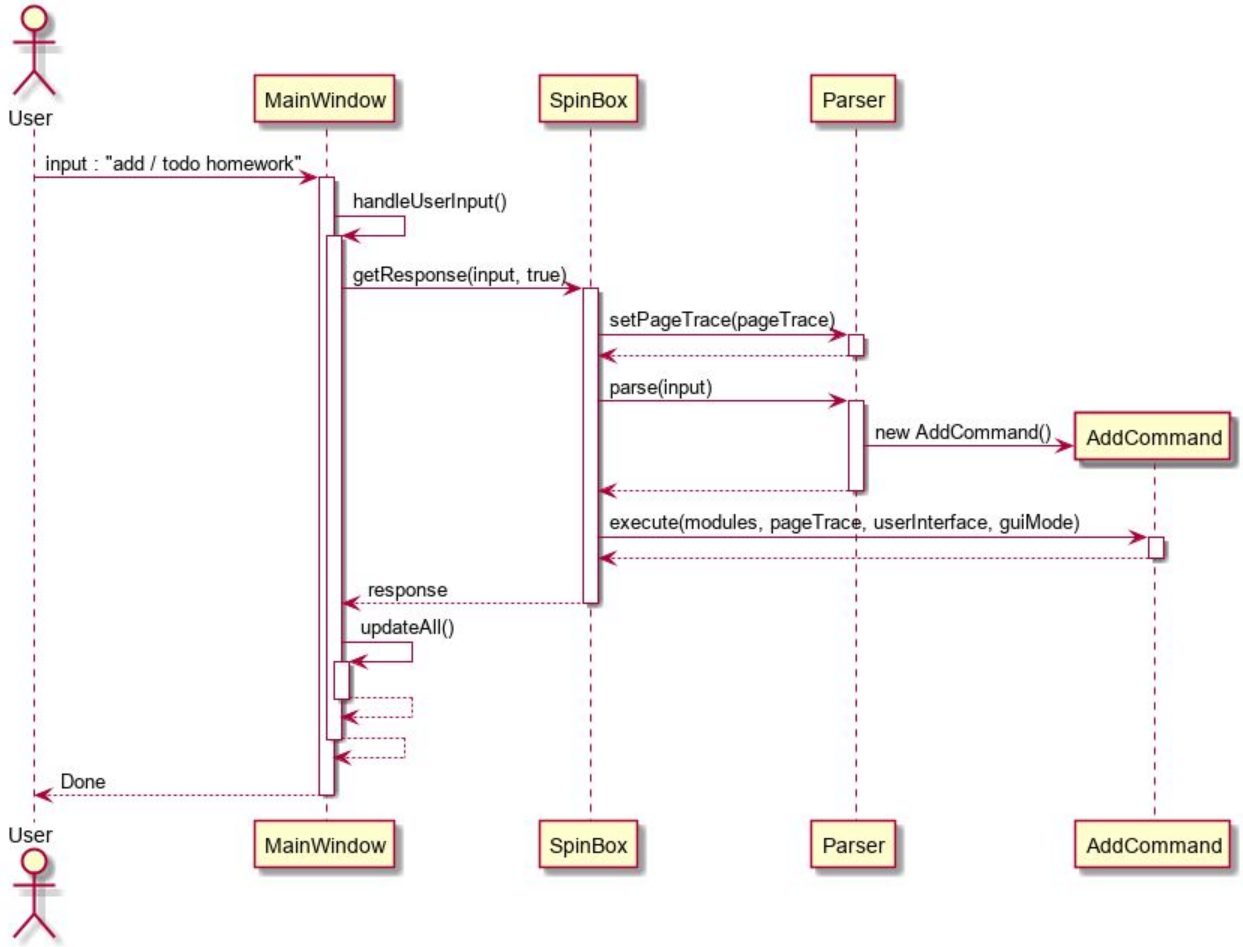

Figure 1.2. Sequential Diagram for Overall Execution

3.2. Gui component

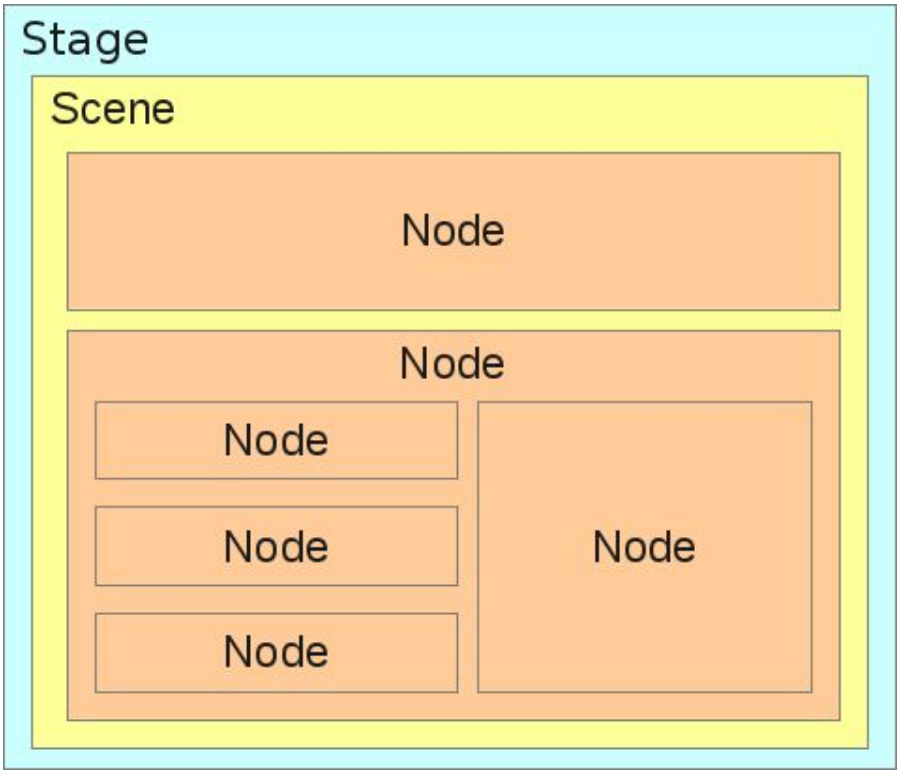

Figure 2.1. Sample JavaFX Node Hierarchy

JavaFX is used to handle the `Gui` component of the program. JavaFX uses node hierarchy to arrange their components ***(Figure 2.1)***. These nodes are contained inside a `Scene` which are contained in a `Stage`.

The `Main` Class in our application sets the `Stage` and `Scene` and sets our `MainWindow` as the root node. All nodes need to be JavaFX components and therefore, `MainWindow` extends a JavaFX component: `GridPane`.

`MainWindow` layout is set by the matching `MainWindow.fxml` files under `src/main/resources/view` folder. Similarly, other `Gui` Classes would have its fxml files in the same folder. In addition, there is also css file if there are any styling to be added to the layout.

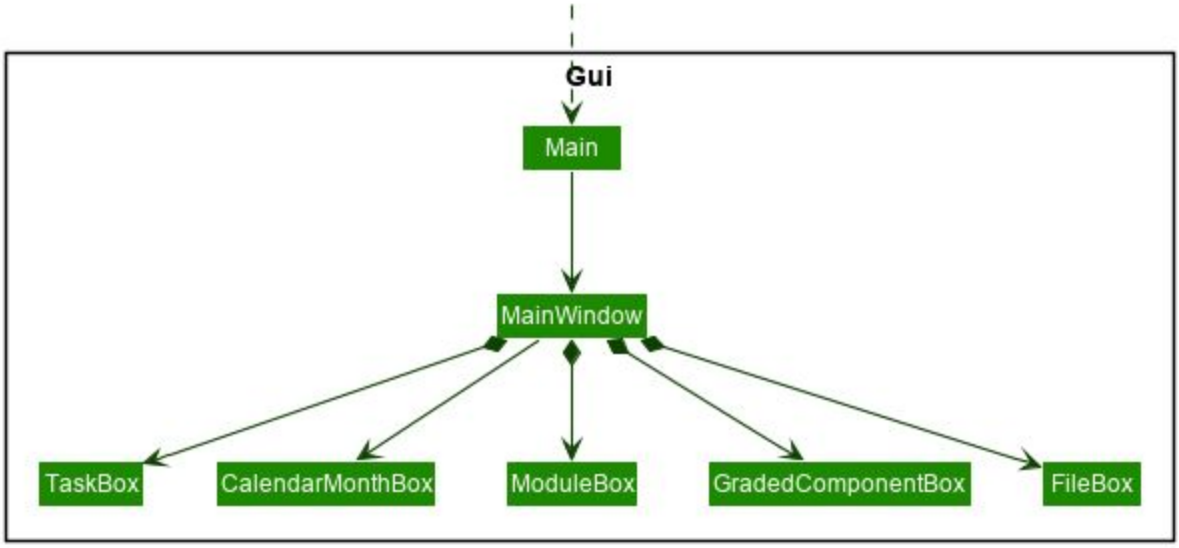

Figure 2.2. Structure of Gui Component

Inside the `MainWindow`, although it contains `Gui` classes ***(Figure 2.2)***, it also contain JavaFX classes which logic, layout and styling are done inside `MainWindow` itself.

This is a sample of the Gui Window with the tab at Main ***(Figure 2.3)***.

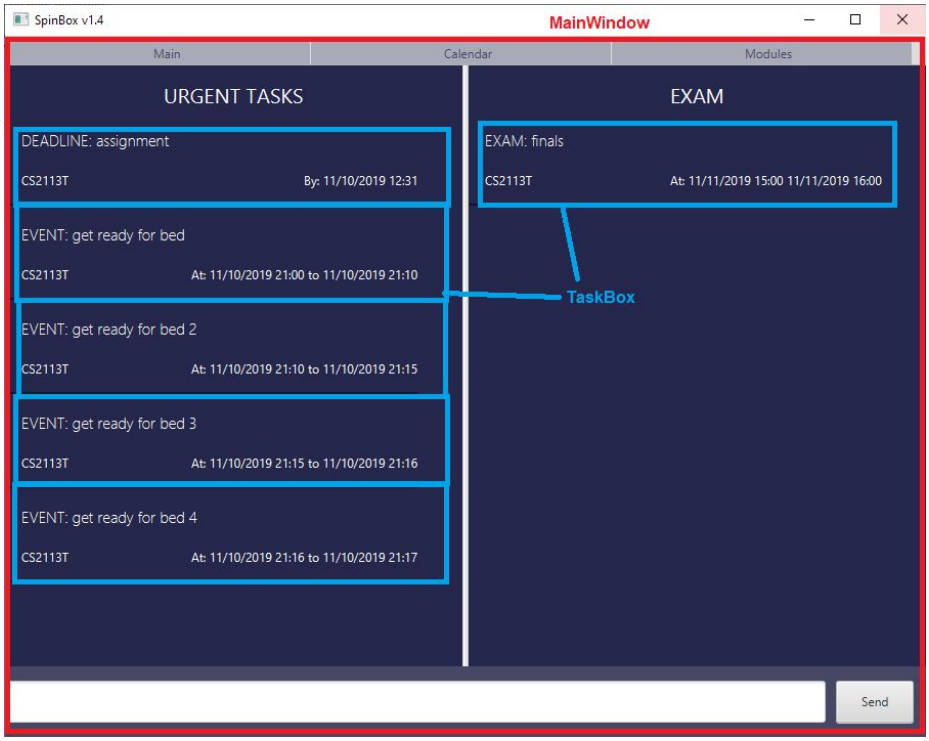

Figure 2.3. Sample of GUI Window

Inside `MainWindow`, examples of its child nodes are JavaFX components are the `TabPane` to swap between tabs, `TextField` for user to write their input and `Button` for the to send the input. `SplitPane` is a child node of `TabPane` and is used to correctly align the two `VBox`. `TaskBox` would populate the `VBox`. Inside, it contains three labels to show the module code, type with description and the date and time.

This list of urgent tasks and exams is generated inside `MainWindow`.

3.3. Command classes

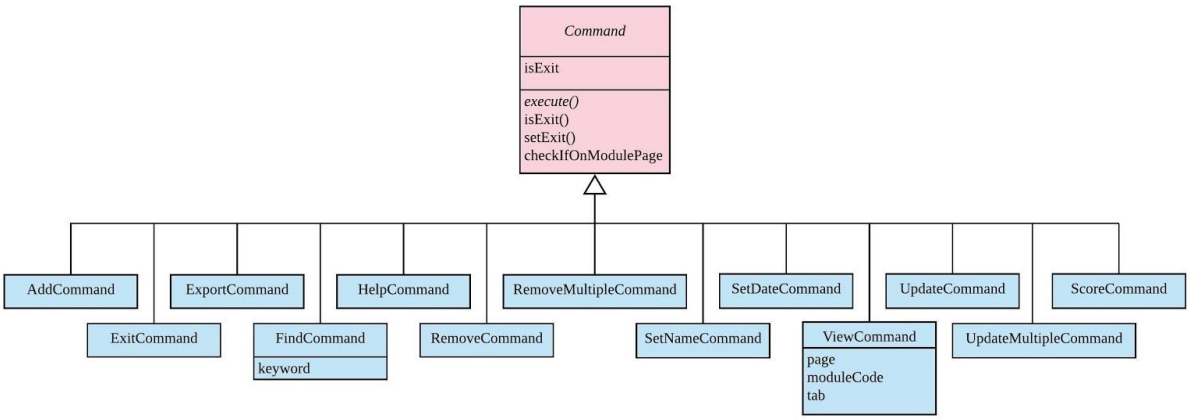

Figure 3. UML Object Diagram for Command Classes

There is an abstract `Command` class. The `execute()` method is abstract as each derived class will have different actions for when the method is called.

The `execute()` is an inherited method as the main program execution calls `execute()` on the parent class `Command` and not the specific derived class as seen in Figure 1, where `Parser` returns a `Command` object to `SpinBox` and `SpinBox` calls `execute()` on the returned `Command` object.

3.4. List classes

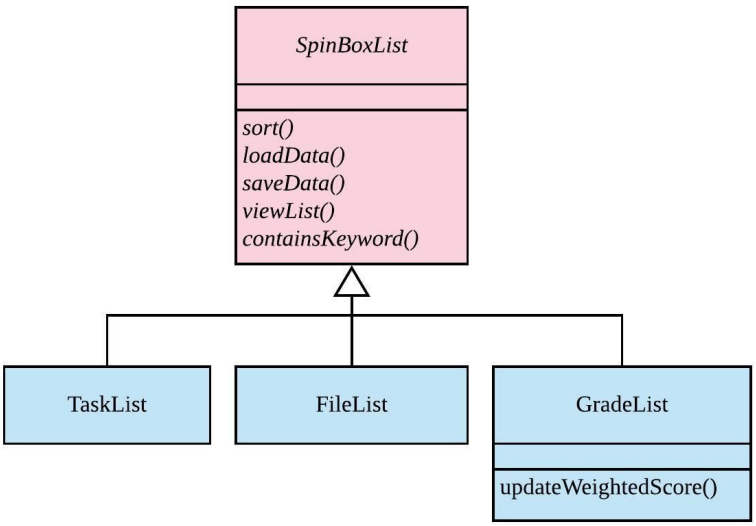

Figure 4. UML Object Diagram for List Classes

There is an abstract generic `SpinBoxList`. It is the parent of:

- `TaskList`
- `FileList`
- `GradeList`

The lists are used to save the different item types. All Lists have:

- `List`
- `parentCode`
- `Storage`

They share fundamental methods that are implemented in such as:

- `add()`
- `remove()`
- `getList()`
- `size()`

These methods are non-abstract and are implemented in the parent class `SpinBoxList`.

However, there are methods that are item dependent, therefore they are declared `abstract` in `SpinBoxList` and are implemented only in the derived class.

3.5. Item classes

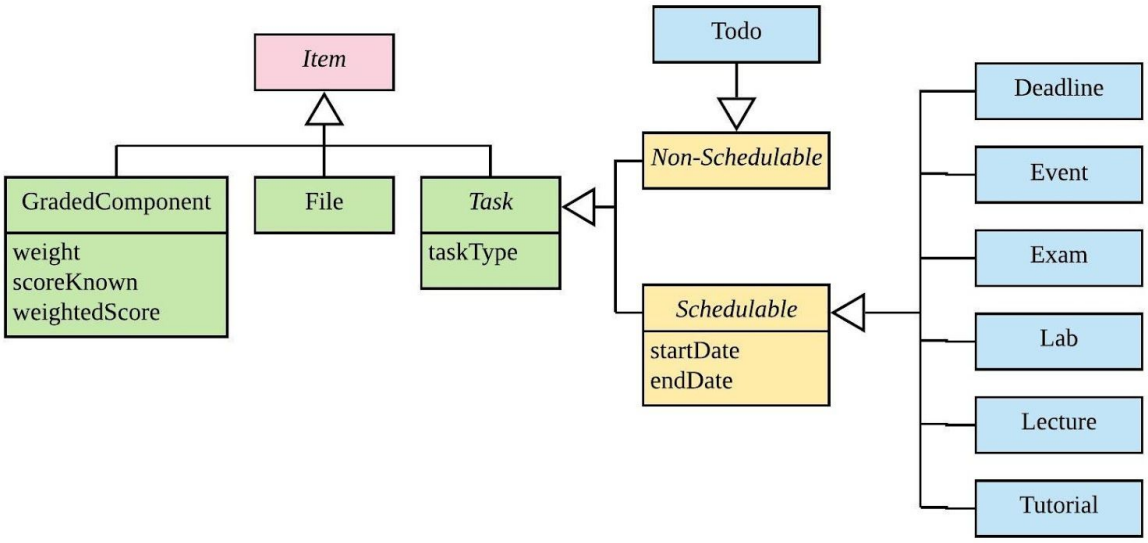

Figure 5. UML Object Diagram for Items

These following classes inherit from the `Item` abstract class:

- `Task`
- `GradedComponent`
- `File`

For `GradedComponents`, there are methods and variables surrounding the weightage of each graded component. These are stored under the `weight`, `scoreKnown` and `weightedScore` variables.

We also create the following subclass to inherit `Task`:

- `Schedulable`
- `NonSchedulable`

Under `NonScheduable`, there is only one subclass which is `Todo`.

For `Schedulable`, there are the following 6 classes that inherits it.

- `Deadline`
- `Event`
- `Exam`
- `Lab`
- `Lecture`
- `Tutorial`

Under `Task`, there is an `enum` called taskType to be able to differentiate between the subclasses. The difference between `Schedulable` class and `Non-Schedulable` class is the two addition DateTime object: startDate and endDate.

4. Implementation

This section provides you with a description of how certain key features of our product are being implemented.

4.1. Intuitive Command Interpreter

4.1.1 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).

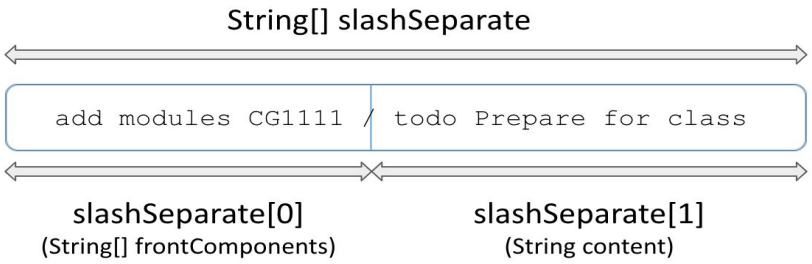

Figure 6. String array slashSeparate

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).

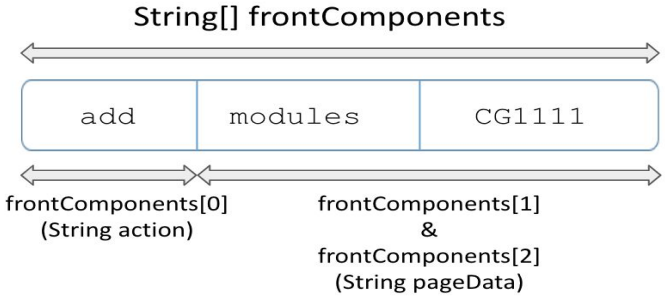

Figure 7. String Array frontComponents

Step 4. `pageData` is now trimmed and set to lowercase before it is passed into the method `commandBuilder` for further processing. `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, given that the input is valid and no exception is thrown. 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).

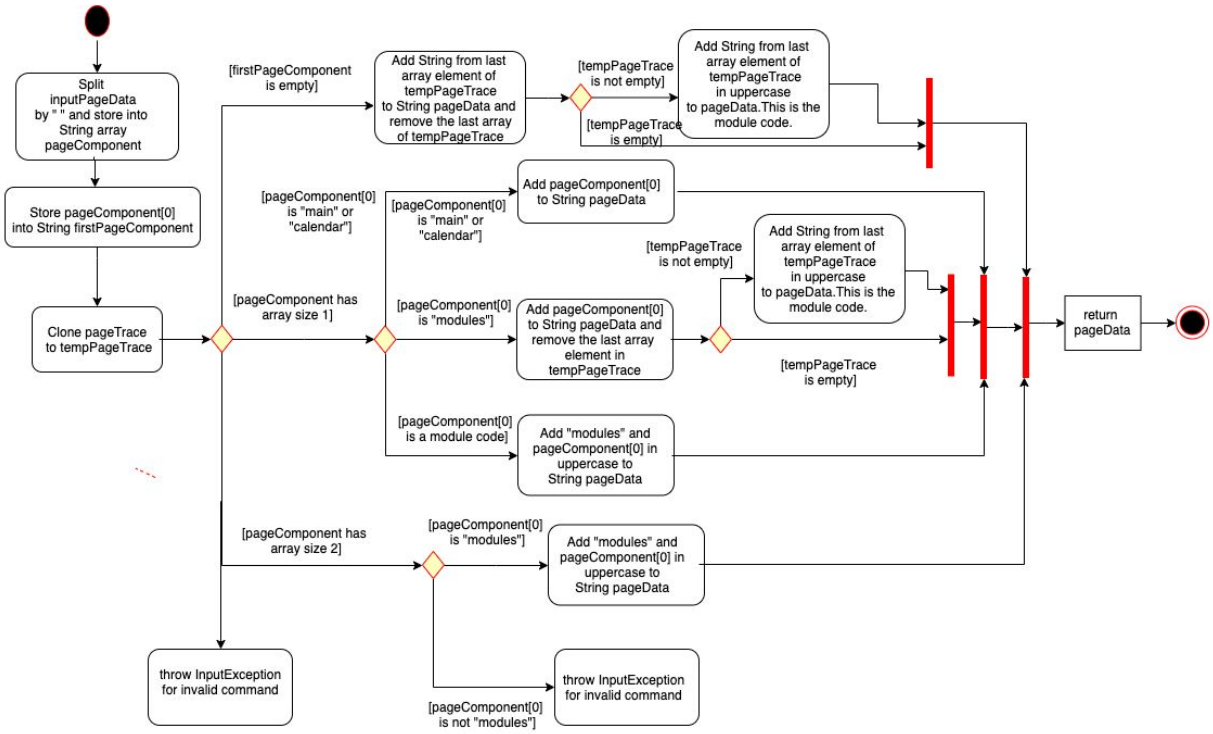

Figure 8. Activity diagram for the method commandBuilder

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`.

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).

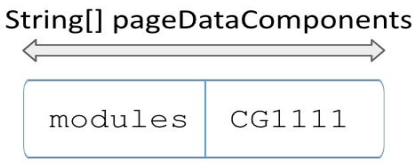

Figure 9. String array pageDataComponents

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 To-do 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).

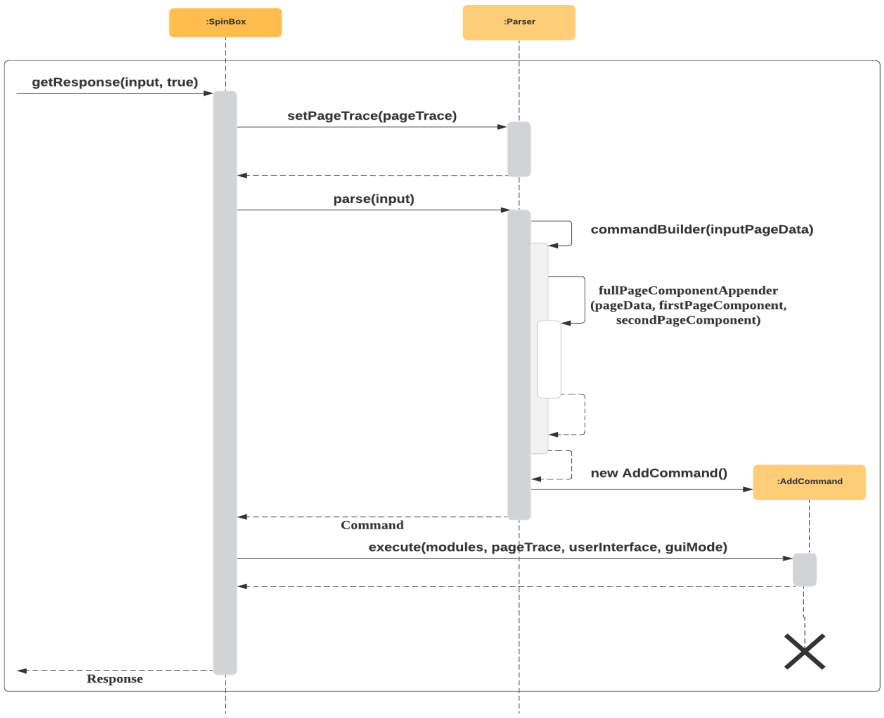

Figure 10. Sequence diagram for the above example

4.1.2 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.

4.2. Hierarchical Data Persistence & Snapshot exports

4.2.1 Implementation

Data-persistence is facilitated by independent instances of the `Storage` and `Exporter` classes, which are designed to be context-agnostic and play the role of functional elements within a larger, highly cohesive component. Through this, greater separation of concerns can be achieved by abstraction of the process of storage and optionally, retrieval of data from files.

Both of these classes inherit from the `DataWriter` class, which sets up the write stream to the physical file on the hard disk.

The file hierarchy is established by attaching a private `Storage` instance within each component. For example, a single `ModuleContainer` instance, will have its own `Storage` to contain the list of modules, with each module's private `FileList`, `TaskList`, `GradeList` and `Notepad` having their own `Storage` instances to store `File`, `Task`, `GradedComponent` and `Note` objects respectively. Each nested instance is provided a path that is prepended with the path of a storage higher up in the hierarchy, with the parent having a directory of `/SpinBoxData`. Due to this, files are created in a nested manner that closely mimics the structure of modules in SpinBox. The resultant file hierarchy is a nested file structure that is similar to the diagram below.

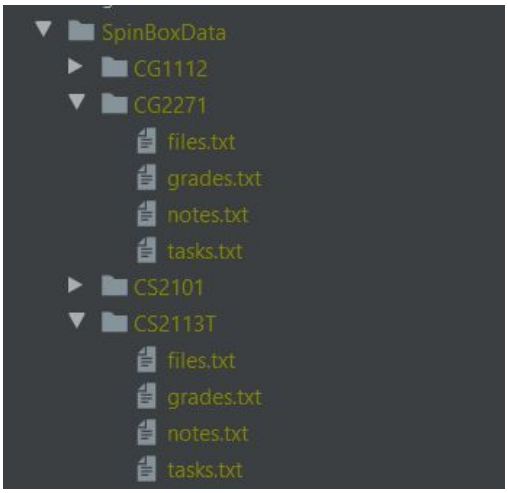

Figure 11: The hierarchy of data stored by SpinBox

The benefits of this is that the advanced user who is familiar with SpinBox can modify the data within in an intuitive manner as they will i) be familiar with the structure through the usage of SpinBox and ii) it closely follows what a human user will do in terms of file organisation by modules.

This section provides an example on how the storage is related to a module's `GradeList` instance, which is essentially a container for `GradedComponents`. The relevant parts of the structure of the program is shown in the below UML class diagram.

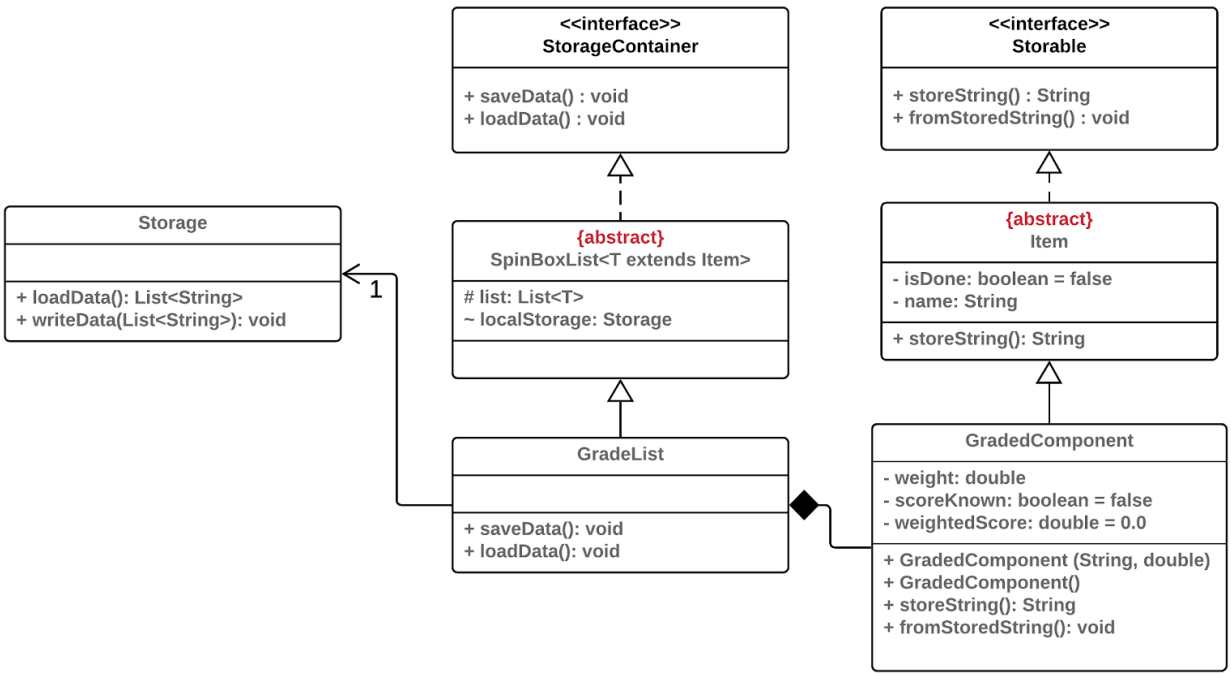

Figure 12: UML class diagram for `Storage` and `GradeList`

As can be seen above, the `Storage` instance is embedded within a `GradeList`. The `GradeList` implements the methods required by the `StorageContainer` interface, such that it can convert the data to/from a `List<String>` for storage. To achieve this conversion, it can call each `GradedComponent`'s methods implemented from the `Storable` interface.

To provide context-agnostic behaviour, `Storage` instances only work with `List<String>` objects as parameters and return values from its methods. Hence, components (usually containers and lists) which make use of a `Storage` instance will have to implement a `StorageContainer` interface, which includes two methods `saveData()` and `loadData()`. Within these methods, conversion of the contained objects to strings and vice-versa is performed. This is facilitated by the objects themselves implementing methods from the `StorableObject` interface.

SpinBox allows for data export to give the users the ability to use SpinBox in conjunction with other specialized productivity applications such as, long term tracking setting reminders or sharing with others, such as Google Keep or Calendar.

Users would want to export their data into English, so that they can naturally use their preferred reminder system in combination with SpinBox to increase their productivity.

The three main types of items - `Task`, `File` and `GradedComponent` implement the `Exportable` interface, which necessitates that they have the `exportString()` method that constructs a human-readable string from their instance variables.

For example, the export feature can export all tasks of a module, as described by the sequence diagram below.

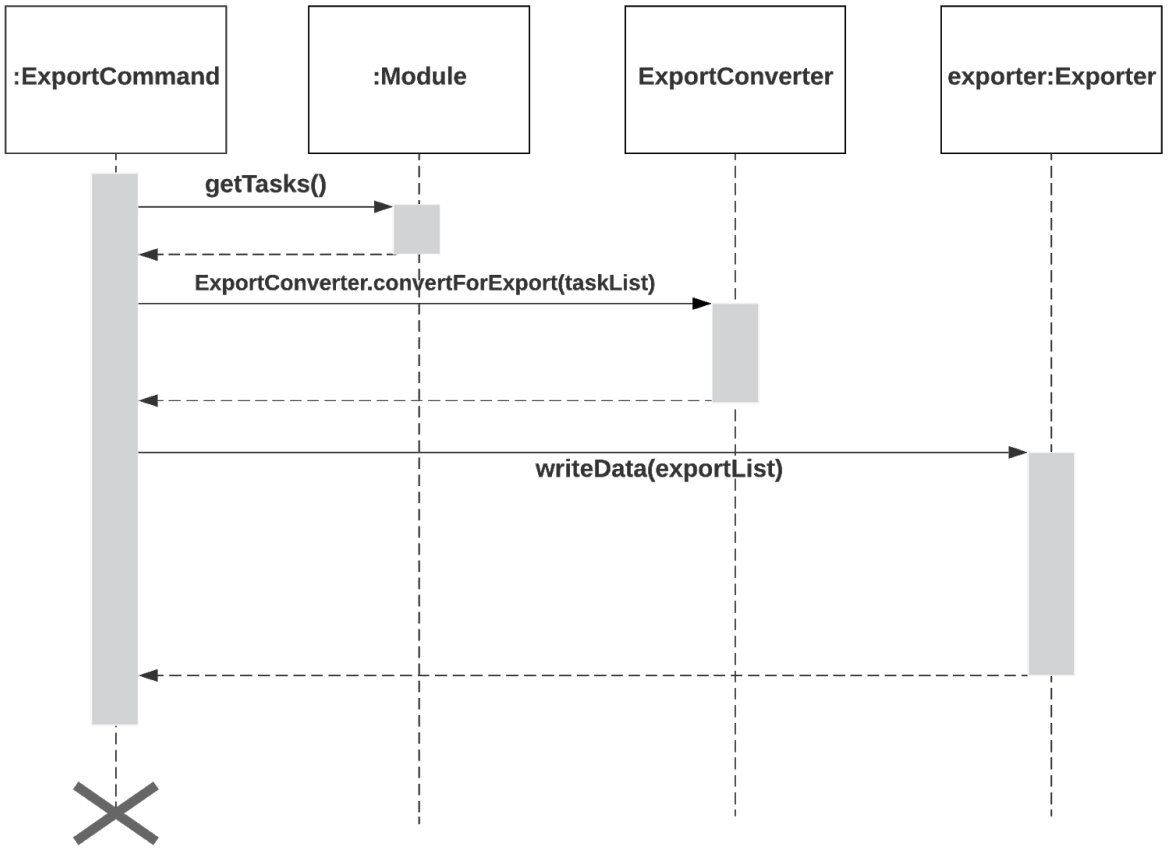

Figure 13: Sequence diagram for exporting task data of a specific module

When a user enters a command for exporting data, the GUI layer passes it to the `Parser`, which creates an instance of an `ExportCommand`. This `ExportCommand` identifies the appropriate `Module` and relevant list and uses the static method of the `ExportConverter` class as a service to convert the list into an `exportList` - a list of strings created from calling each input `Exportable`'s `exportString` method.

This is then passed to the `Exporter` instance for writing to a physical file on the hard disk under "SpinBox/exports".

4.2.2 Design Considerations

Aspect	Alternative 1 (Chosen)	Alternative 2
Distribution of data	One separate instance/file per container. Pros: ● No concurrent I/O issues (simultaneous reads + writes) if multi-threaded ● Uniform data type - no requirement to filter through irrelevant entries ● Enables hierarchical storage hierarchy for intuitive editing experience for advanced users ● Single file corruption will not affect all data ● Higher cohesion ● Simple to implement and reusable in a customised manner within any Container implementing StorageContainer Cons: ● Storage must be generalized to a certain extent, currently a requirement of List<String> ● Requires external conversion to adhere to the Law of Demeter ● Time/space overhead due to multiple files	One instance that stores into one file. Pros: ● Space efficient Cons: ● High coupling ● Filtering must be done on entries ● More difficult to users to edit and remain corruption-free

Function set for storage	Set/Reset + Load (Current choice) Pros: ● Easy to implement ● Appropriate as each list is scoped small-enough that it is not expected to store too many items ● Can extract out & reuse for efficient export Cons: ● Processing overhead	CRUD Pros: ● Mirrors actions performed on Containers ● Performance-friendly Cons: ● Requires additional state or stored information (e.g. indexes) ● Slightly harder implementation

4.3 GradedComponents

In a typical module, the assessments are spread out through the semester and have varying weightages. For a student, these can quickly get overwhelming to manually keep track of, and SpinBox makes this process pain-free through the implementation of graded components.

Given below is a typical use case for the `GradeList` component, and the sequence of events when marking a `GradedComponent` as complete, after being made aware of the grade obtained.

Step 1: The user starts up SpinBox to score a written report worth 20% under CS2999. SpinBox scans through the SpinBoxData folder and populates its variables in memory.

Step 2: The user views a module's grades through **view CS2999 / grades**.

Step 3: The user executes **score / 2 marks: 87/100** to update their score into SpinBox.

The technical workflow can be described through the activity diagram below, which starts at the GUI layer, after which the command is parsed to identify the module and component.

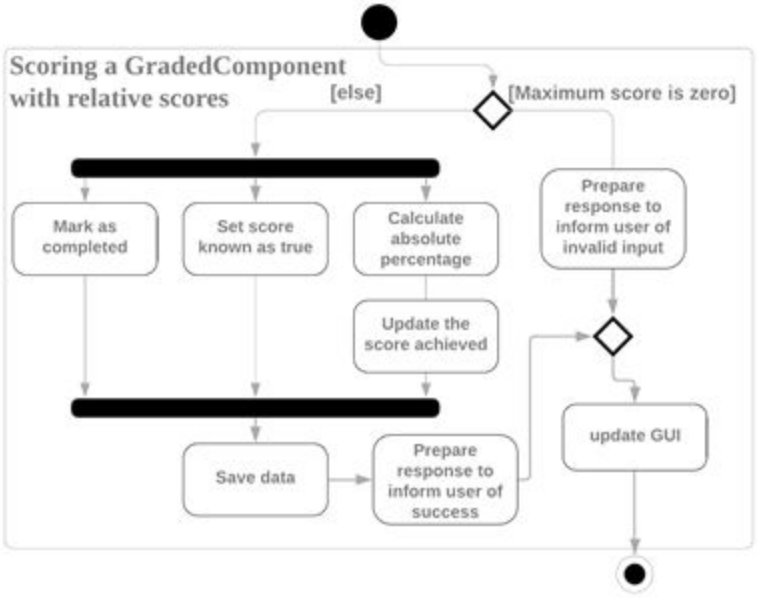

Figure 14: Sequence diagram for scoring a graded component

Within the actual code, uncommon failures, such as erroneous user input, are handled through a hierarchy of exceptions, which are caught or propagated upwards to the `SpinBox` instance, where the message to send back to the user is constructed with the help of the `Ui`.

4.4 DateTime and Calendar

4.4.1 Implementation

The date and time for each `Schedulable Task` is being keep tracked by `DateTime`. In this class, it has a private `java.util.Date`. For this class, there are mainly three ways to construct it. The first way with `DateTime(Date)`, the second with `DateTime(String)` and the third with `DateTime(String, Int)`. For construction with `String` instead of `Date` object, which can be passed easily into the private variable, the `String` containing the `DateTime` would be converted into a `Date` object via third party library called `Natty`.

DateTime implements the following operations on top of trivial functions:

- `DateTime#before(DateTime)/DateTime#after(DateTime)/DateTime#equal(DateTime)` — Return a boolean stating whether this `DateTime` object's date and time is before/after/equal to the `DateTime` object passed
- `DateTime#getStartOfTheWeek()/DateTime#getEndOfTheWeek()` — Return another `DateTime` object with date set as the start/end of the week relative to this `DateTime.`
- `DateTime#getStartOfTheMonth()/DateTime#getEndOfTheMonth()` — Return another `DateTime` object with date set as the start/end of the week relative to this `DateTime.`
- `DateTime#toString()` — Return `DateTime` in `String` version of `MM/dd/yyyy HH:mm` which can be similarly used to create another `DateTime` object.

The first sets of functions are useful to see if a pair of `DateTime` objects overlaps with another pair of `DateTime` objects. Currently it is utilised by all `Schedulable Task` to see if it overlaps with any other tasks.

The second and third sets of functions are useful to get the range of days in a week or month which can be paired with the first sets of functions to effectively filter out `Schedulable` tasks that are in that range. This will be utilised by `Calendar` to populate the UI to allow the user to see different events and deadlines that are scheduled in that week or month.

4.5. Searching within items

4.5.1 Implementation

4.5.1.1 Overall Implementation

This feature allows the user to search for specific items that contains a keyword in the name of the item. It is implemented in the `findCommand` class.

To run the command, the user inputs:

Format: `find / <itemType> <keyword>`

Example: `find / file tutorial`

To search for all files in the current module that contains the word ‘tutorial’ in the filename.

Below is the execution process from the input through the whole program:

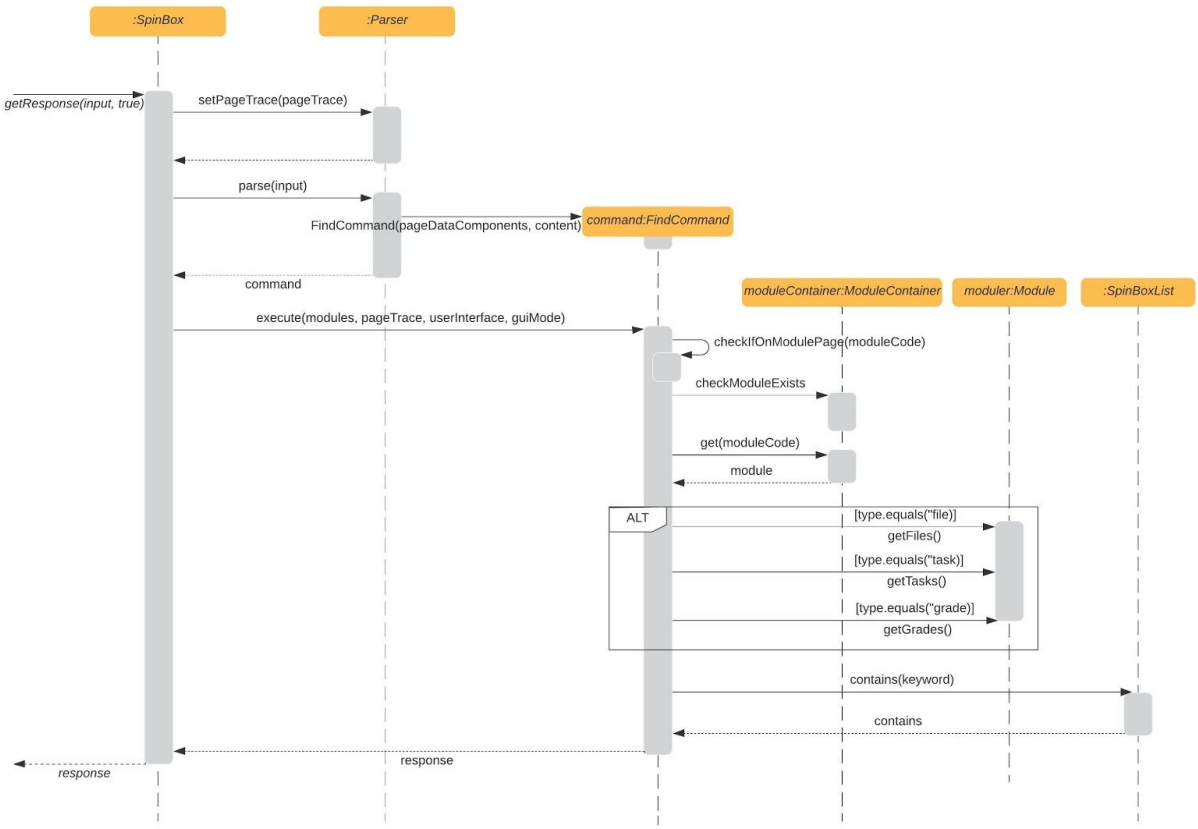

Figure 15. Sequence Diagram for Find Command

Explanation of the process:

1. When the user first enters the input, it is executed the same way as most other commands. However, the command `find` will let `Parser` return a command of type `FindCommand` to `SpinBox`.

2. When the `FindCommand` instance is created, `pageDataComponents` and `content` is passed as parameters to the constructor.

3. The constructor will firstly check whether there is an existing context, specifically if `pageDataComponents` has more than 1 element, since it is known that the second element will always be the `moduleCode`. The type of item to be searched, (`file`, `task`, `grade`), is also extracted.

- This tells the program which module to access for the list
- If there is no `moduleCode` found, then throw an `InputException`.

4. When the `execute` method is called on `command:FindCommand`, the first action is to extract the keyword by splitting the content using the first space and assigning `keyword` to the second element.

5. Afterwards, it calls the method `checkIfOnModulePage` that is defined in the parent class `Command` to check that the `moduleCode` is not null.

6. Then the `checkModuleExists` method from `moduleContainer` is called to check that the `moduleCode` given is an existing module.

7. Then if it is true, `findCommand` accesses the module by calling `get(moduleCode)`, which returns the module required.

8. Depending on the type of item to be searched, `findCommand` will either call `getFiles`, `getTasks`, or `getGrades` on the `module` to access the correct list.

9. Afterwards, the `containsKeyword` method is called on the list, which will return a list of items that contains the keyword in their name in string format, which will be returned in turn to `SpinBox` to be outputted.

4.5.1.2 containsKeyword Implementation

The `containsKeyword` method is declared abstract in the parent class and is implemented in the derived classes. `containsKeyword` requires `keyword` as a parameter and works by iterating through each element in the list and checking whether the name of the item contains the keyword.

The code to check the file item is:

```
file.getName().toLowerCase().contains(keyword.toLowerCase())
```

4.5.2 Design Considerations

4.5.2.1 Correct Keyword is Extracted

To make sure that `keyword` is not restricted to only words and can include sentences with spaces, the `split` method called on content is limited to two elements as the first one will be item type and the second will be the rest of the content.

Additionally, to account for whitespace in front or behind the keyword, `trim` is called on the keyword.

4.5.2.2 Sorted list of items with keyword found

To ensure that the list of items that contain `keyword` is sorted, the `containsKeyword` method is declared abstract in `SpinBoxList`. This is because the custom comparator created in each derived class is needed as different lists have different sorting methods.

Even though the method is iterating through its list, which is sorted, `containsKeyword` still sorts it again to ensure that the outputted list is sorted even if the list it is iterating through is not.

4.6 Viewing different pages

4.6.1 Implementation

4.6.1.1 Overview

`SpinBox` has different pages that can be accessed. The hierarchy of the pages is given as:

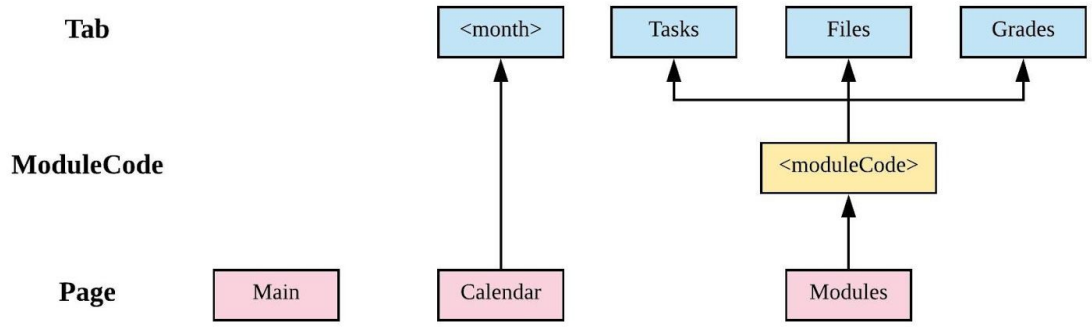

Figure 16. Diagram on the hierarchy of the view pages

- For `SpinBox` to be on the specific module page, it needs to firstly be on the `modules` main page.
- For `SpinBox` to be on a tab page, it needs to be on both the `modules` main page and a `moduleCode` page.

To change pages, the user uses the `ViewCommand`. `ViewCommand` has two parts, the `constructor` and `execute` methods.

4.6.1.2 Constructor

The activity diagram for the `constructor` is given below:

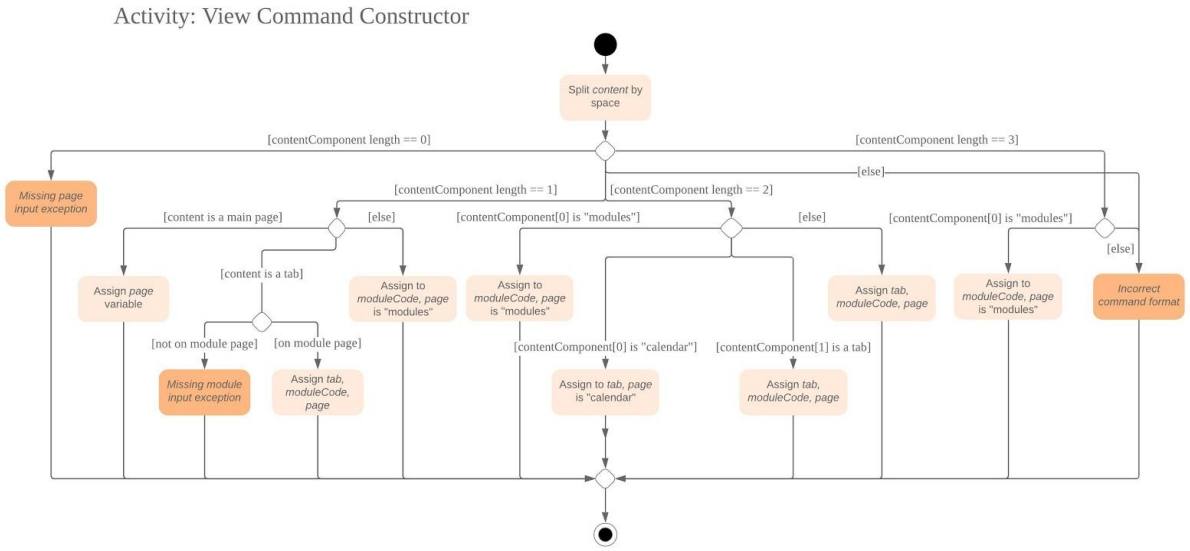

Figure 17. Activity Diagram for ViewCommand Constructor

The `ViewCommand` constructor takes `pageDataComponents` and `content` as input. The purpose of the constructor is to assign `page`, `moduleCode`, and `tab` is applicable, and leave it as `null` if it is not.

`contentComponents` length is 0:

- This means that there is no page to go to so throw `InputException`.

`contentComponents` length is 1:

1. It is a `mainPage`
    - Check if it is `main`, `calendar`, or `modules` .
2. It is a `tab`
    - Check if it is `tasks`, `files`, or `grades` .
3. It is a `moduleCode`
    - It cannot be verified as the program gives flexibility of `moduleCode` naming.
    - Check if valid in `execute`.

`contentComponents` length is 2:

- The only options if there are two pages according to Figure X. is `modules <moduleCode>` or `<moduleCode> <tab>` or `calendar <month>`.

1. If first component is `modules`
    - Assign `page` to `modules`, and `moduleCode` to `contentComponents[1]`
    - `moduleCode` cannot be verified as the program gives flexibility of `moduleCode` naming. Check in `execute`.
2. If first component is `calendar`
    - Assign `page` to `calendar`, and `tab` to `contentComponents[1]`
    - `tab` cannot be verified, will be checked in `execute`.
3. If second component is `tab`
    - Assign `page` to `modules`, `moduleCode` to `contentComponents[0]` and `tab` to `contentComponents[1]`

`contentComponents` length is 3:

- The only option is `modules <moduleCode> <tab>`

4.6.1.3 Execute

The activity diagram for the `execute` is given below:

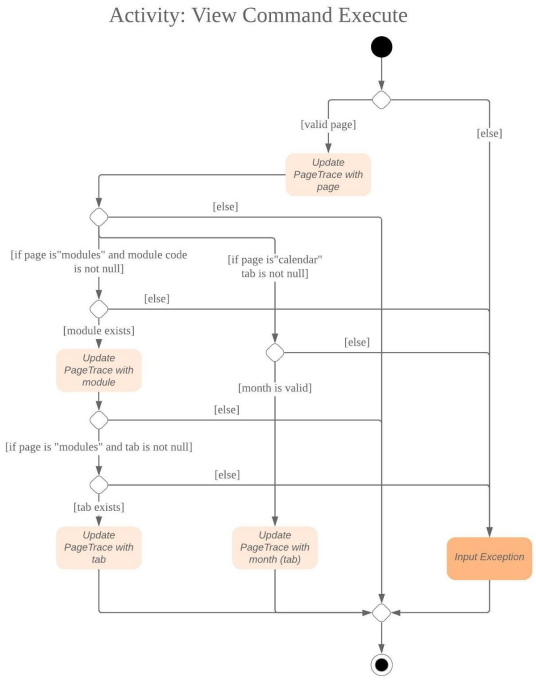

Figure 18. Activity Diagram for ViewCommand Execute

When executing, `ViewCommand` saves the old `pageTrace` and keeps adding on to a new `pageTrace` when `page`, `moduleCode`, or `tab` is not `null`.

- The default `tab` for a specific module page is `tasks`.
- `execute` checks whether `moduleCode` exists in `moduleContainer`
- `execute` checks whether `tab` is a valid month if `page` is `calendar` and `tab` is not `null`

4.6.2 Considerations

4.6.2.1 Hierarchy of pages

The hierarchy of the different pages were decided based on what seems the most intuitive.

- The three tabs for each module is related as they all belong to the same module, hence this should be reflected in the hierarchy.
- Each module specific page should be related as they are on the same “level”.
- The months in `calendar` can be viewed as different tabs.

Therefore, having a “flat” hierarchy is disadvantageous as there is no relation between the pages. “flat” hierarchy means every page is equal and independent of each other.

The current hierarchy is intuitive because

- There are 3 main pages which is reflected in the layer `page`.
- Each module specific page builds on the main page `modules`, hence, they should be all one layer above `modules` and dependent on `modules`.
- All `tab` should be of the same layer but they are related to a `moduleCode`

4.6.2.2 Implied main page is modules

It is unnecessary to type `modules` if a user is going directly to a module specific page or tab. Therefore, the `constructor` automatically assigns `page` to be `modules` if first component of `content` is `moduleCode` or `tab`.

5. Testing

5.1. Running Tests

There are two ways to run tests.

💡	The most reliable way to run tests is the 3rd one. The first two methods might fail some GUI tests due to platform/resolution-specific idiosyncrasies.

Method 1: Using IntelliJ JUnit test runner

- To run all tests, right-click on the `src/test/java` folder and choose `Run 'All Tests'`
- To run a subset of tests, you can right-click on a test package, test class, or a test and choose `Run 'ABC'`

Method 2: Using Gradle

- Open a console and run the command `gradlew clean allTests` (Mac/Linux: `./gradlew clean allTests`)

ℹ	The most reliable way to run tests is the 3rd one. The first two methods might fail some GUI tests due to platform/resolution-specific idiosyncrasies.

5.2. Types of tests

We have two types of tests:

1. Integration Tests - These are tests that are checking the integration of multiple code units (those code units are assumed to be working).

   e.g. `java.integration.FileListIntegrationTest`

2. Unit Tests - These are tests targeting the lowest level methods/classes.

    e.g. `java.unit.TaskListUnitTest`

6. Dev Ops

6.1. Build Automation

See UsingGradle.adoc to learn how to use Gradle for build automation.

6.2. Continuous Integration

We use Travis CI and AppVeyor to perform *Continuous Integration* on our projects. See UsingTravis.adoc and UsingAppVeyor.adoc for more details.

6.3. Coverage Reporting

We use Coveralls to track the code coverage of our projects. See UsingCoveralls.adoc for more details.

6.4. Making a Release

Here are the steps to create a new release.

1. Update the version number in `MainApp.java`.
2. Generate a JAR file using Gradle.
3. Tag the repo with the version number. e.g. `v0.1`
4. Create a new release using GitHub and upload the JAR file you created.

6.5. Managing Dependencies

A project often depends on third-party libraries. Managing these *dependencies* can be automated using Gradle. For example, Gradle can download the dependencies automatically, which is better than these alternatives:

1. Include those libraries in the repo (this bloats the repo size)
2. Require developers to download those libraries manually (this creates extra work for developers)

Appendix

Appendix A: Product Scope

SpinBox is for 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 grades breakdown for every module.

Target user profile:

- NUS students who are seeking assistance in managing school life
  - handling deadlines
  - keeping track of deliverables and important dates
  - monitoring module updates
  - viewing grades breakdown for every module
- NUS students who prefer desktop apps over other types
- NUS students who can type fast
- NUS students who prefer typing over mouse input
- NUS students who are are reasonably comfortable using CLI apps

Value proposition: Manage task tracking faster than a typical mouse/GUI driven app

Appendix B: User Stories

Priorities:

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

Priority	As a …	I want to …	So that …
* * *	NUS Student	be able to view a calendar of events or task for the upcoming week or month	I can easily keep track of them
* * *	NUS Student	be able categorize items by module	I can view items relevant to just one module at a time

* * *	NUS Student	be able to add my own actionable items related to a particular module	I can keep track of everything related to that module.
* * *	NUS Student	be able to delete multiple tasks from my task list	it will be more convenient to manage my tasks.
* * *	NUS Student	be able to sort my tasks into different priority levels	I will not miss out on any high priority tasks
* * *	NUS Student	be able to mark my task as undone from the done state	I can revert my changes if I unintentionally mark a task as done
* * *	NUS Student	be able to set the grade breakdown for each module at the start of the semester	I can refer back to it easily without having to look at the introduction lecture slides
* * *	NUS Student	be able to refer to the grade breakdown for each module	I can prioritize my efforts correctly throughout the semester
* * *	NUS Student	be able to add in my own custom notes for each module	I can keep track of miscellaneous items that cannot be placed under a structured category like deadlines (i.e. email the lecturer to clarify something)

* * *	NUS Student	be able to view a list of deliverables with their deadlines by module	I can keep track of what is due for the module
* * *	NUS Student	be able to view upcoming deliverables for each module in a sorted manner	I can quickly figure out what is due next
* * *	NUS Student	be able to add and remove any module in the modules tab	I can use this application for my subsequent semesters when I take new modules
* * *	NUS Student	be able to view the exam dates for my different modules at the main page and the module page	I will not miss out on these important details during the exam period
* * *	NUS Student	be able to view the aggregate upcoming deadlines on the main page	I can be aware of my priorities across all modules
* * *	NUS Student	be able to mark files that I have downloaded	I can easily see which files still need to be downloaded
* * *	NUS Student	be able to edit my tasks or files description after I have added them into the application	I can ensure that the information I have is the latest one
* *	NUS Student	be able to have a checklist of graded components that I can tick off once I have completed a component	I can focus on the things that are yet to come
* *	NUS Student	be able to view all the module announcements at one look	I will not miss out on any important announcements

* *	NUS Student	have different colour codes for each module	I can easily distinguish the modules
* *	NUS Student	be able to add my own actionable items related to a particular module	I can keep track of everything related to that module.
* *	NUS Student	be able to reschedule anything on my schedule	I can deal with things that are pushed back or tasks which are preempted by a higher priority task
* *	NUS Student	be able to view the location details of my exam venue	I can plan my journey to the exam venue in advance
* *	NUS Student	have a countdown timer which indicates the number of days from my exam	I can better pace myself when I am doing revision for my different modules
* *	NUS Student	have my tutorials, lectures and user-defined events to be colour-coded	I can better identify what is the type which my events fall under
* *	NUS Student	see the upcoming deadlines for the upcoming day, week and month	I can plan accordingly.
* *	NUS Student	be able to add files that have been uploaded on LumiNUS	I can keep track of the files available
*	NUS Student	be able to update my grades breakdown with the scores I've obtained, if made known to me	I can track how well I'm doing in the module throughout the semester

*	NUS Student	be able to export the grades I've gotten for a completed module to a permanent file	I can reflect on my shortcomings and make lifestyle changes for future semesters
*	NUS Student	generate a simple text file of all my deadlines	I can print the upcoming deadlines.
*	NUS Student	generate a simple text file of all my tasks	I can view them externally.
*	NUS Student	generate a simple text file to see the schedule for the week	I know what are my classes, events, deadlines, and exams for the week and I have a clear overview of the week
*	NUS Student	have my tutorials, lectures and user-defined events to be colour-coded	I can better identify what is the type which my events fall under
*	NUS Student	see my overall CAP from the modules I have already cleared	I am clear on how well I am currently doing

Appendix C: Use Cases

(For all use cases below, the System is the `SpinBox` and the Actor is the `NUS Student User`, unless specified otherwise)

Use case 1: View a high level summary of upcoming items

1. User opens the application
2. SpinBox displays the main page, consisting of most urgent deadlines and modules with exam dates

3. User navigates to calendar
4. SpinBox displays a calendar of tasks as well as timetable items
5. User navigates to modules
6. SpinBox shows the user a list of modules added

Use case 2: Add module

MSS

1. User enters input to add module (exam date, venue, name, code, grade breakdown)
2. SpinBox acknowledges that the module has been added successfully
3. User looks at the list of modules
4. SpinBox shows all modules, including the one added

Extensions

- 1a. SpinBox detects an error in the User input
- 1b. SpinBox alerts the user of the failure to add module
- 1c. User re-enters input in a different manner
- Steps 1a- 1c are repeated until input is suitable
- Use case resumes from step 2

Use case 3: View module details

Preconditions: User has launched application

1. User navigates to modules
2. SpinBox shows a list of modules
3. User selects a particular module
4. SpinBox shows a tabbed view of [tasks, files, grades] and a free-text notes box
5. User can continue navigating within the module as they prefer

Use case 4: Add task

Preconditions: User has added at least 1 module by following Use case 2

MSS

1. User follows UC03 to enter a module view
2. SpinBox populates and displays module view with existing data, if any
3. User enters input to add a deadline
4. SpinBox acknowledges addition of deadline
5. User navigates to module -> tasks tab
6. SpinBox shows deadline as part of task list
7. User navigates to calendar
8. SpinBox shows deadline in the relevant calendar view
9. User navigates to main
10. SpinBox shows deadline as part of pending tasks

Extensions

- 3a. User enters incorrect input 3b. SpinBox alerts user of failure to add deadline
3c. User re-enters input
Steps 3a-3c repeat until input is suitable
Use case resumes from 4
- 10a. SpinBox does not show deadline if it is not within the most urgent tasks

Use case 5: Complete task

Preconditions: User has added at least 1 task by following Use case 5

MSS

1. User follows UC03 to enter a module view
2. SpinBox populates and displays module view with existing data, if any
3. User enters input to mark a task as complete
4. SpinBox acknowledges task completion
5. User navigates to module -> tasks tab
6. SpinBox no longer shows task as part of task list
7. User navigates to calendar
8. SpinBox no longer shows task in the relevant calendar view

9. User navigates to main
10. SpinBox no longer shows task as part of pending tasks

Extensions

- 3a. User enters incorrect input
- 3b. SpinBox alerts user to failure to complete task
- 3c. User re-enters input
- Steps 3a -3c repeat until suitable input is entered
- Use case resumes from step 4

Appendix D: Non Functional Requirements

1. Should work on any mainstream OS as long as it has Java `11` or above installed.
2. Store data in a manner that is easily understandable for advanced users - use an appropriately named folder/file hierarchy system
3. Storage must be sufficiently quick, with deferring of operations till there is a need to perform it until required
4. Acknowledgement for user input must be helpful and guide them towards MSS
5. Acknowledgement for user input must be extremely fast

Appendix E: Glossary

Mainstream OS

Windows, Linux, Unix, OS-X

Appendix F: Instructions for Manual Testing

Given below are instructions to test the app manually.

ℹ	These instructions only provide a starting point for testers to work on; testers are expected to do more *exploratory* testing.

F.1. Launch and Shutdown

1. Initial launch
   1.1. Download the jar file and copy into an empty folder
   1.2. Double-click the jar file
   Expected: Shows the empty GUI with a red, bolded message within the input box
   1.3. Type in ‘populate’ into the input box to populate the application with sample data

2. Exiting the application
   2.1. Type ‘bye’ into the input box to exit the application

F.2. Adding and removing a module

1. Add a module
   1.1. Prerequisite: The module does not already exist within SpinBox, which can be verified under the modules tab, accessible via `view / modules`
   1.2. Test case: `add / module CG1111 EPP1`
   1.3. Expected: The module is added successfully and can be viewed under the modules tab.

2. Removing a module
   2.1. Prerequisite: The module must exist within SpinBox, which can be verified under the modules tab, accessible via `view / modules`
   2.2. Test case: `remove / module CG1111`
   2.3. Expected: The module is removed, and is no longer visible under the modules tab.

F.3. Interacting with a Module

1. View a module
   1.1. Prerequisite: The module does exsit within SpinBox, which can be verified under the modules tab, accessible via `view / modules`
   1.2. Test case: `view / CG2271`
   1.3. Expected: The module is successfully viewed with the Tasks subtab open

2. Viewing a specific sub-tab
    2.1. Possible sub-tabs: One of [`tasks, files, grades`]
    2.2. Prerequisite: To use the short version of the command, you must already be viewing the module
    2.3. Test case 1 (short): `view / tasks`
    2.4. Test case 2 (full): `view / CG2271 tasks`
    2.5. Expected: The subtab is shown, with the notepad always shown on the left side of the screen.

F.4. Adding or removing a task/file/grade/note

1. Adding a task/file/grade/note
    1.1. Prerequisite: To use the short version of the command, you must already be viewing the module
    1.2. Prerequisite: The module must currently exist within SpinBox
    1.3. Type `help / add` within the input box to see arguments for each
    1.4. Test case 1 (short): `add / grade <arguments>...`
    1.5. Test case 2 (full): `add CG1111 / grade <arguments>...`
    1.6.
2. Remove a task/file/grade/note
    2.1. Prerequisite: To use the short version of the command, you must already be viewing the module
    2.2. Prerequisite: The module must currently exist within SpinBox
    2.3. Type `help / remove` within the input box to see arguments for each
    2.4. Test case 1 (short): `remove / grade <arguments>...`
    2.5. Test case 2 (full): `remove CG1111 / grade <arguments>...`

F.5. Find a task/file/grade

1. Finding keyword in tasks
    1.1. Prerequisite: To use the short version of the command, you must already be viewing the module
    1.2. Prerequisite: The module must currently exist within SpinBox
    1.3. Prerequisite: There are tasks named “a”, “b”, “a b” in the module
    1.4. Type `help / add` within the input box to see arguments for each
    1.5. Test case 1 (short): `find / task a`
    1.6. Test case 2 (full): `find CG1111 / task a`
    1.7. Test case 1 (short): `find / task a b`

1.8. Test case 1 (short): `find / task a`

2. Finding keyword in files
   2.1. Prerequisite: To use the short version of the command, you must already be viewing the module
   2.2. Prerequisite: The module must currently exist within SpinBox
   2.3. Prerequisite: There are files named “a”, “b”, “a b” in the module
   2.4. Type `help / add` within the input box to see arguments for each
   2.5. Test case 1 (short): `find / file a`
   2.6. Test case 2 (full): `find CG1111 / file a`
   2.7. Test case 1 (short): `find / file a b`
   2.8. Test case 1 (short): `find / file a`

3. Finding keyword in files
   3.1. Prerequisite: To use the short version of the command, you must already be viewing the module
   3.2. Prerequisite: The module must currently exist within SpinBox
   3.3. Prerequisite: There are graded components named “a”, “b”, “a b” in the module
   3.4. Type `help / add` within the input box to see arguments for each
   3.5. Test case 1 (short): `find / grade a`
   3.6. Test case 2 (full): `find CG1111 / grade a`
   3.7. Test case 1 (short): `find / grade a b`
   3.8. Test case 1 (short): `find / grade a`

F.6. Changing pages

1. View main page
   1.1. Type `help / view` within the input box to see view command usage
   1.2. Test case 1: `view / main`

2. View calendar page
   2.1. Type `help / view` within the input box to see view command usage
   2.2. Test case 1: `view / calendar`
   2.3. Test case 2: `view / calendar 01/2020`

3. View module page
   3.1. Type `help / add` within the input box to see view command usage
   3.2. Test case 1: `view / modules`

4. View specific module page
    4.1. Prerequisite: CG1111 is a module that has been added already.
    4.2. Type `help / view` within the input box to see view command usage
    4.3. Test case 1 (short): `view / CG1111`
    4.4. Test case 2 (full): `view / modules CG1111`

5. View specific tab
    5.1. Prerequisite: CG1111 is a module that has been added already.
    5.2. Prerequisite: To use the short version of the command, you must already be viewing the module
    5.3. Type `help / view` within the input box to see view command usage
    5.4. Test case 1 (short): `view / tasks`
    5.5. Test case 2 (full): `view / CG1111 tasks`
    5.6. Test case 3 (max): `view / modules CG1111 tasks`
    5.7. Test case 4 (short): `view / files`
    5.8. Test case 5 (short): `view / grades`

F.7. Exporting Data

1. Exporting grades
    1.1. Prerequisite: CG1111 is a module that has been added already.
    1.2. Prerequisite: To use the short version of the command, you must already be viewing the module
    1.3. Test case 1(Short): `export / grades`
    1.4. Test case 2 (Full): `export CG1111 / grades`
    1.5. Expected: Exported File containing grades data inside SpinBoxData/exports

2. Exporting files
    2.1. Prerequisite: CG1111 is a module that has been added already.
    2.2. Prerequisite: To use the short version of the command, you must already be viewing the module
    2.3. Test case 1(Short): `export / files`
    2.4. Test case 2 (Full): `export CG1111 / files`
    2.5. Expected: Exported File containing files data inside SpinBoxData/exports

3. Exporting deadlines
    3.1. Test case : export / deadlines
    3.2. Expected: Exported File inside SpinBoxData/exports

F.8. Scoring graded components

1. Scoring graded components with relative score
    a. Prerequisite 1: Have a graded component within the current module with a known index (refer to GUI - number next to GradedComponent)
    b. Prerequisite 2: To use the short version of the command, you must already be viewing the module
    c. Test case 1 (Short): `score / 1 marks: 27/30`
    d. Test case 2 (Full): `score CG2271 / 1 marks: 27/30`

F.9. Other commands

1. Please type `help` within the SpinBox input box to see more advanced commands and explore features such as set / update / remove multiple and scoring graded components.
    a. We wish you a happy testing experience!