SPINBOX

TANAT TIPPINYU - PROJECT PROFILE PAGE

OVERVIEW

SpinBox is a program aimed for NUS students who are seeking assistance in managing their school life. It is optimized for those who prefer to work with a Command Line Interface (CLI) while still having the benefits of a Graphical User Interface (GUI).

SUMMARY OF CONTRIBUTIONS

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

Enhancement added: Search items that contain a specific keyword

- What it does: Search through items of a specific type for names that contain the keyword given
- Justification: Students may want to find all items related to a topic, for examples finding all files labelled tutorial
- Highlights: the keyword is flexible, it works with starting and trailing whitespace. It also works for sentences, so your keyword can be a statement. Lastly, the outputted list is still sorted.

Enhancement added: View different pages

- What it does: Allows users to change page
- Justification: Students will want to view the page for different modules and the different tabs for the modules.

Code Contributed: [RepoSense], [FindCommand], [ViewCommand]

Other Contributions:

- Made the structure for storing the items. Implemented the generic `SpinBoxList`, and its derived classes `TaskList`, `FileList`, and `GradeList`. Created custom comparator for sorting.
- Was in charge of creating issues and assigning them to ensure that the project is on track.
- Played a major role in the architecture of the whole program. Made sure that the execution logic is intuitive and the classes are inherited and derived correctly to reduce duplication and improve the intuitive relationship between classes.

3.2. CHANGE TO THE MAIN TAB : `VIEW / MAIN`

Default active tab when the application is started. Displays a list of the five most urgent pending tasks across all modules on the left side of the screen and a list of modules on the right (Figure 1).
Format: `view / main`

3.3. VIEWING THE CALENDAR TAB : `VIEW / CALENDAR`

Displays the calendar in the calendar tab populated using task data across all modules. The calendar only displays tasks that have a start and end date (a To do or deadline will not show up on the calendar). The current month is displayed by default, unless the full command is used.
Format: `view / calendar` or `view / calendar [MM/YYYY]`

An expected output is shown below.

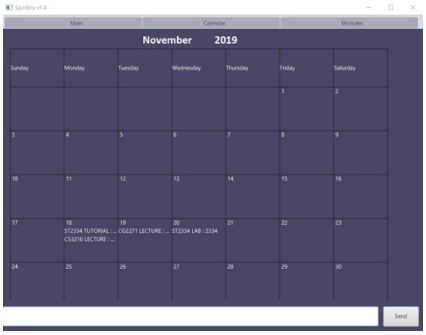

Figure 2: The calendar tab, displaying the schedule for November 2019

3.4. VIEWING THE MODULES TAB : `VIEW / MODULES`

Displays the high-level modules tab, which contains a list of the user's added modules. Each module appears as a clickable button, which is one of the ways to access the details of the module.
Format: `view / modules`

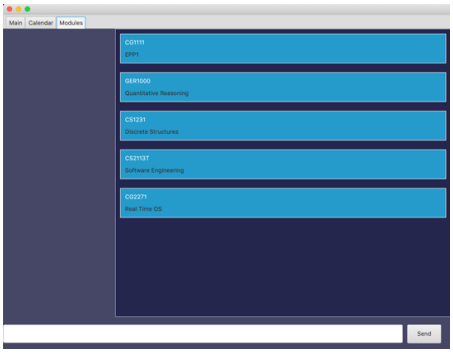

Figure 3: Modules tab, which contains a list of clickable modules

3.7. LOCATING MODULE COMPONENT BY NAME : FIND

Finds module component with names that contain any of the given keywords within the currently selected module. The output will be a list of items that contains the specified keyword. Alternatively, use the full version if the module is not currently selected.

Format: `find / [ITEM_TYPE] [KEYWORD]` or
`find [MODULE_CODE] / [ITEM_TYPE] [KEYWORD]`

- Possible `[ITEM_TYPE]`: task, file, grade
- The search is case insensitive. e.g `Work` will match `work`
- Only the name is searched.
- Keyword can contain spaces. E.g. `Tutorial 1` is a possible keyword.

Examples:

`find CG1112 / file tutorial` (returns all files in CG1112 containing the word 'tutorial' in its filename)

- `find / task exam` (returns all tasks containing 'exam' in the module that the user is currently on)

CONTRIBUTIONS TO THE DEVELOPER GUIDE

2.4. LISTCLASSES

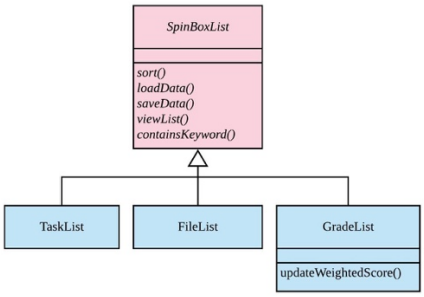

Figure 4. UML Object Diagram for List Classes

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.

3.4. SEARCHING WITHIN ITEMS

3.4.1 Implementation

3.4.1.1 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:

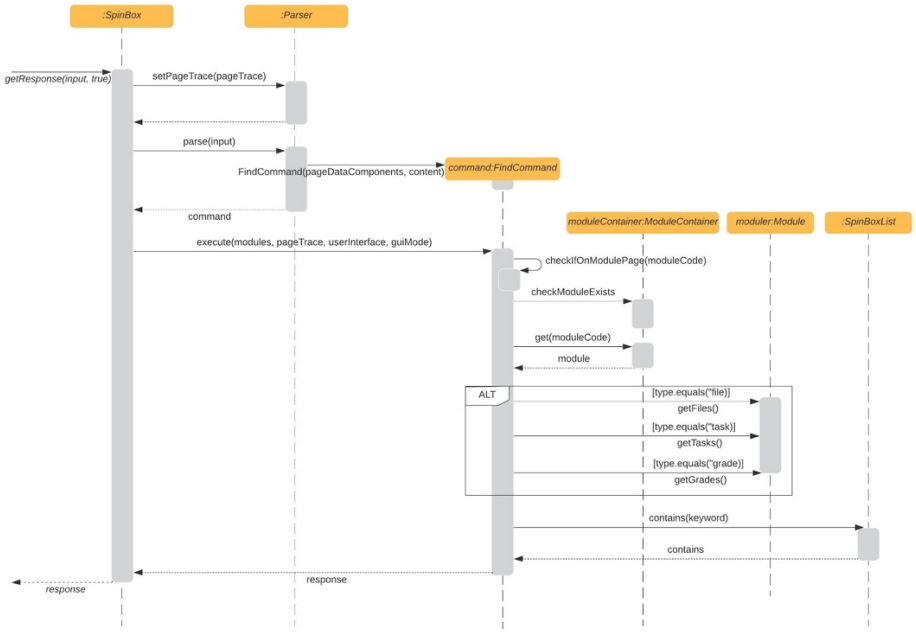

Figure 15. Sequence Diagram for Find Command

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.

*****THE REST OF SECTION 3.4 IN THE DEVELOPER GUIDE IS NOT INCLUDED IN THIS DOCUMENT*****

3.5. VIEWING DIFFERENT PAGES

3.5.1 Implementation

3.5.1.1 Overview

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

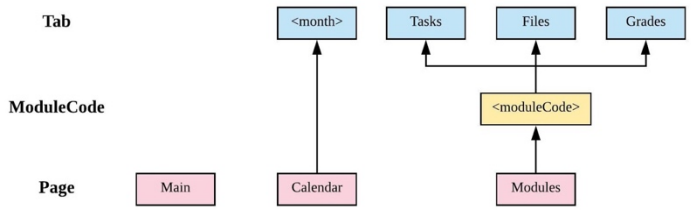

Figure 16. Diagram on the hierarchy of the view pages

- 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.

3.5.1.2 Constructor

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

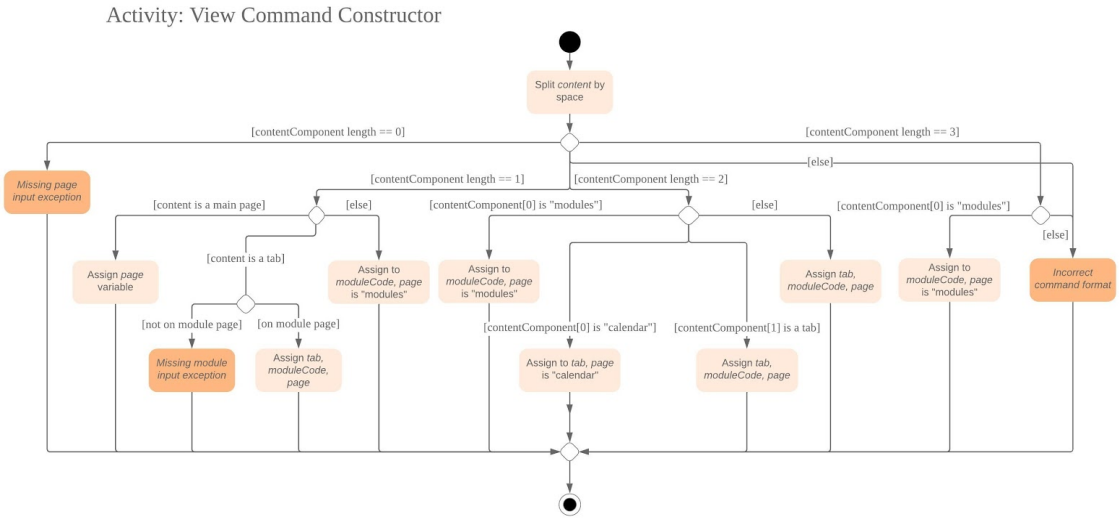

Figure 17. Activity Diagram for ViewCommand Constructor

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 16. 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. Checked 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]`

4. `contentComponents` length is 3:
    - The only option is `modules <moduleCode> <tab>`

3.5.1.3 Execute

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

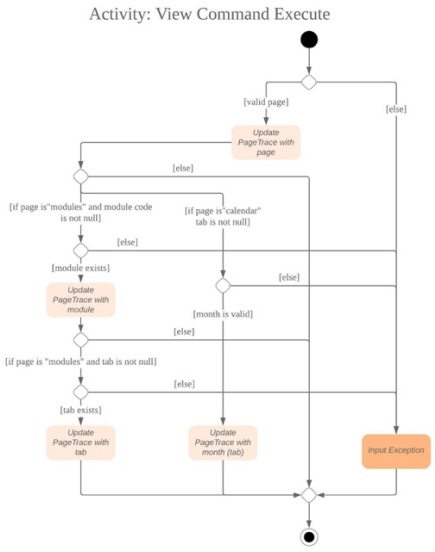

Figure 18. Activity Diagram for ViewCommand Execute

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`

3.5.2 Considerations

3.5.2.1 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`

3.5.2.2 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`.