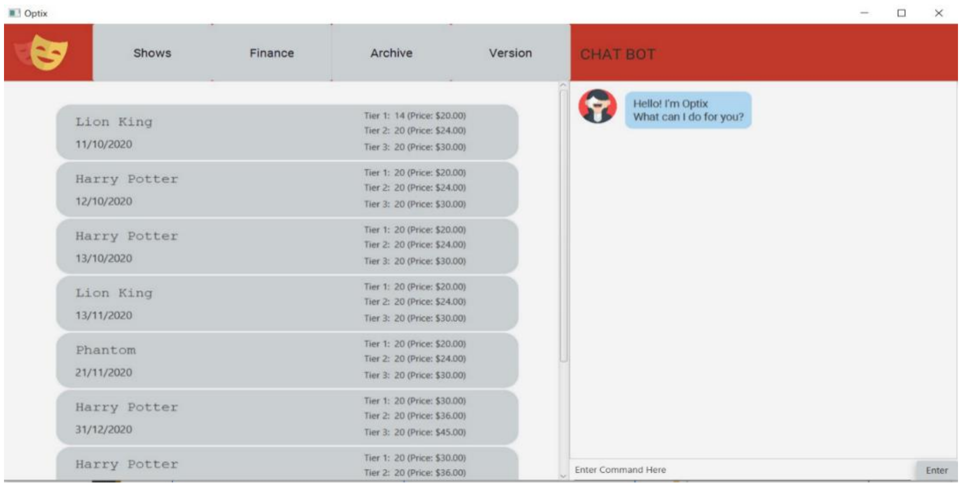
Kennedy Oung – Project Portfolio for Optix

About the Project

My team of 3 computer engineering students and I were tasked with enhancing a basic Task Manager application for our Software Engineering (CS2113T) project. We decided to morph into a show and ticketing management system called Optix. This application would be intended for use on desktop computer where users would interface the application primarily through the command line. The application is meant for students who intend to organize and sell tickets/ seats for events. A possible extension of this application would be to modify it for use by concert hall/ cinema management companies.

This is a view of our prototype:



Lion King
11/10/2020
Tier 1: 14 (Price: $20.00)
Tier 2: 20 (Price: $24.00)
Tier 3: 20 (Price: $30.00)

Harry Potter
12/10/2020
Tier 1: 20 (Price: $20.00)
Tier 2: 20 (Price: $24.00)
Tier 3: 20 (Price: $30.00)

Harry Potter
13/10/2020
Tier 1: 20 (Price: $20.00)
Tier 2: 20 (Price: $24.00)
Tier 3: 20 (Price: $30.00)

Lion King
13/11/2020
Tier 1: 20 (Price: $20.00)
Tier 2: 20 (Price: $24.00)
Tier 3: 20 (Price: $30.00)

Phantom
21/11/2020
Tier 1: 20 (Price: $20.00)
Tier 2: 20 (Price: $24.00)
Tier 3: 20 (Price: $30.00)

Harry Potter
31/12/2020
Tier 1: 20 (Price: $30.00)
Tier 2: 20 (Price: $36.00)
Tier 3: 20 (Price: $45.00)

Harry Potter
Tier 1: 20 (Price: $30.00)
Tier 2: 20 (Price: $36.00)

Hello! I'm Optix
What can I do for you?

Figure 1 The graphical user interface of Optix

Note the following symbols and formatting used in this document:

- This symbol indicates important information.

`delete` • A grey highlight in a different bolded font indicates that this is a command that can be input into the command line and executed

CommandAliasMap • A grey highlight with blue text indicates a component, class or object in the architecture of the application.

Summary of Contributions

My role was to design some Commands that execute instructions, and the Parser that interprets user input. Two prominent features that I developed was the *Command Aliasing* and *Delete Command* features.

A secondary role that I took on was to analyse potential faults in the code and devise test cases for Commands and Models. I also managed detail logging of the application.

The following sections illustrate my contributions in the code base, as well as the user and developer guides for these enhancements.

Enhancement 1

I added the ability to add aliases for commands and remove unwanted aliases. Additionally, I provided the ability to reset the alias settings to the default settings.

What it does	• The **`add-alias`** command allows users to add an alias for a command. • Users may remove the alias using the **`remove-alias`** command. • If users decide to revert all their alias modifications, they can use the **`reset-alias`** command • Users can use the **`list-alias`** command to list all current aliases.
Justification	• The application is command line based and a significant amount of time is spent typing. The **`add-alias`** command enables users to replace command keywords with shorter abbreviations, hence decreasing time spent typing. • If users changed their mind about adding the alias or made a mistake, they can use the **`remove-alias`** command to delete the alias. • If users decide to undo all of their alias related changes, the **`reset-alias`** command can be used to reset all alias-related modifications. • If users cannot remember all the aliases, they can use the **`list-alias`** command to view all current aliases.
Highlights	Command aliasing works with existing commands and can be extended to future commands with just small adaptations to the code. This follows the *Open-Closed Principle* proposed by Bertrand Meyer[1].
Code contributed	Link to the parser class : [parser] Link to the delete command: [delete] Link to the add-alias command: [add-alias] Link to the remove-alias command: [remove-alias] Link to the list-alias command: [list-alias]

[1] The Open-Closed principle proposes code should be written to be open for extension but closed for modification.

Enhancement 2

I added the ability to delete unwanted shows from the ShowList. This is an important tool that enables users to organize their ShowList and reduce cluttering.

What it does	The **`delete`** command allows users to remove unwanted shows from the ShowMap.
Justification	Users may accidentally add unwanted shows and leaving them there would clutter up the Graphical User Interface(GUI). The **`delete`** command would avoid this problem and would boost user experience. Version 1: Use 2 separate commands for single deletions and mass deletions • **`Delete-one`** command would let users delete a show with a specified showName, on a specified showDate. • **`Delete-all`** command would let users delete all shows with the specified showName. Version 2: • **`Delete`** command would let users delete show according to the specified showName and showDate. This command accepts multiple showDates, allowing for multiple deletions by date.
Highlights	I wrote and fine-tuned Version 1 of the **`delete`** command. After some testing, we(the developer team) found that having 2 separate commands that handled deletion was unintuitive for the user. We decided to refactor the old code to Version 2 afterwards, which my teammate Brian was in charge of. It should be noted that the base code for Version 1 was used in building the code for Version 2.
Code contributed	Version 1: Pull Request #12

Other Contributions:

- Enhancement to Existing Features: Reposense
  - Wrote additional tests for existing features to increase coverage from 60% to 70 %. (Pull Request #66)
- Project Management:
  - Leveraged CheckStyle plugin on Gradle to standardize code structure and ensure that code meets industry coding standards. (Pull Request #76, #91)
  - Utilized Travis Continuous Integration (CI) plugin to simplify code testing and bug finding/ fixing(Pull Request #69).
  - Enabled logging of key information to aid in debugging of code for future developers.(Pull Request #141).
- Code Contributed:
  - Regularly added new features to improve application functionality (RepoSense Dashboard)
- Documentation:
  - Made cosmetic improvements to About Us page(Commit e9e32ce)
  - Designed UML diagrams to illustrate classes, and states of the application at specific instances

Contributions to the User Guide

The user guide is an instructional document for users to understand the features in the application. In addition to making cosmetic improvements to the User Guide, I updated the User Guide to include instructions for some enhancements. The following is an excerpt from the ***Optix User Guide***, showing additions I made for the Command Aliasing features.

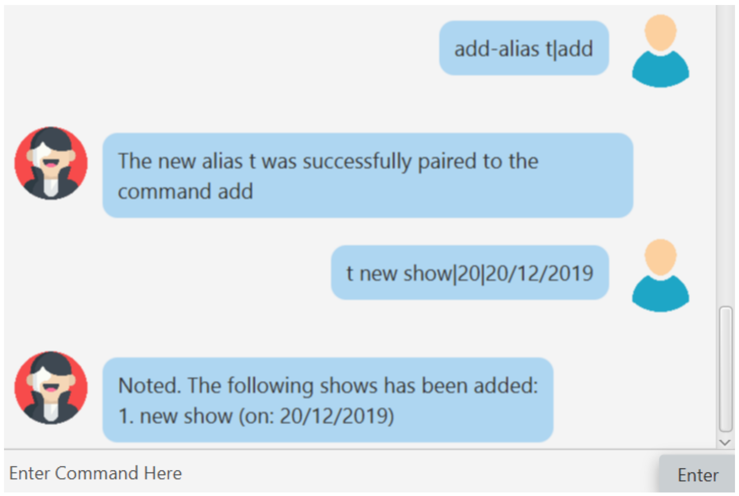
3.5.1 Add new alias: `add-alias`

As a user, you can give a Command an alternate name for easy access. You can add them with **`add-alias`** to an existing command. After adding the alias, it is immediately available for use!
Format: **`add-alias ALIAS|COMMAND`**

- Adds a new alias for `COMMAND`
- The `ALIAS` must not already be in use.
- The `ALIAS` must not be the name of a `COMMAND`.

Examples:

Give the command **`add`** an alias of 't':
`add-alias t|add`


add-alias t|add

The new alias t was successfully paired to the command add

t new show|20|20/12/2019

Noted. The following shows has been added:
1. new show (on: 20/12/2019)

Assigning an alias that is already in use is not allowed.
`add-alias t|delete` would not work if **`'t'`** is paired to another command already.

Excerpt 1: Documentation of the add-alias command from the Optix User Guide

As can be seen from the excerpt, images were liberally used to provide visual aid. This is because learning the software can be a daunting task for users. For users unfamiliar with command-line interfaces, learning the concept of aliasing can be especially difficult. These images would allow users to observe and compare the expected result with the actual output as they follow the instructions step-by-step.

Additionally, all the invalid output was clearly listed and elaborated upon. The intention behind doing so is for users to understand the rules behind using the aliasing function. This will help them better understand the program logic and expedite their learning process.

Contributions to the Developer Guide

The following section shows my additions to the Developer Guide for the `delete` feature.

4.3 Delete Show Feature

Allows users to delete shows from the shows ShowMap.

4.3.1 Implementation

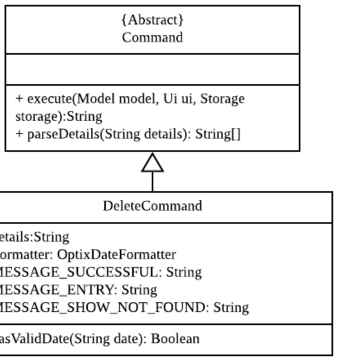
It is executed by the DeleteCommand, which is extended from the abstract class Command, and is stored under the Command package. Additionally, it implements the following operations based on the user input:

- DeleteCommand#hasValidDate -- checks if the input date is of a valid format
- Model#containsKey -- Checks for key in ShowMap.
- Model#hasSameName -- Checks for the existence of the show for the specified showDate in ShowMap.
- Model#deleteShow -- Removes the show from ShowMap.

Given below is an example usage scenario and how the `delete` mechanism behaves at each step.

Step 1

The user inputs `delete Phantom of the Opera|5/5/2020|6/5/2020`, with the intention to delete shows dated on 5th May 2020 and 6th May 2020. The DeleteCommand is initialised with “Phantom of the Opera|5/5/2020|6/5/2020” as the details attribute. The details string is parsed into the show name, and the individual dates.

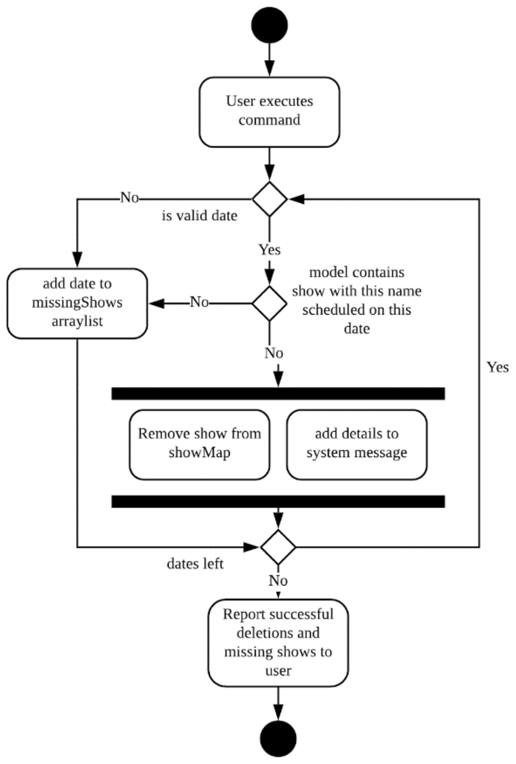

{Abstract}
Command
+ execute(Model model, Ui ui, Storage storage):String
+ parseDetails(String details): String[]
DeleteCommand
- details:String
- formatter: OptixDateFormatter
- MESSAGE_SUCCESSFUL: String
- MESSAGE_ENTRY: String
- MESSAGE_SHOW_NOT_FOUND: String
- hasValidDate(String date): Boolean

Step 2

DeleteCommand iterates through the dates. It formats and verifies them through OptixDateFormatter#hasValidDate(String date) to ensure the given dates are valid(that the date follows the format DD/MM/YYYY). LocalDate instances are generated from these date strings.

Some steps of the Delete Show feature were omitted for brevity.

The activity diagram below illustrates the logic process of the DeleteCommand:



User executes command
No
is valid date
Yes
model contains show with this name scheduled on this date
add date to missingShows arraylist
No
No
Yes
Remove show from showMap
add details to system message
dates left
No
Report successful deletions and missing shows to user

The following code snippet highlights the key logic of the DeleteCommand.

```
int counter = 1;

for (String showDate : showDates) {
   String date = showDate.trim();

   if (!hasValidDate(date)) {
       missingShows.add(date);
       continue;
   }
```

```
    LocalDate showLocalDate = formatter.toLocalDate(date);

    if (model.containsKey(showLocalDate) &&
model.hasSameName(showLocalDate, showName)) {
        model.deleteShow(showLocalDate);
        message.append(String.format(MESSAGE_ENTRY, counter,
showName, date));
        counter++;
    } else {
        missingShows.add(date);
    }
}
```

4.2.2 Design Considerations

Aspect: How delete works

- **Alternative 1:** Splitting the delete feature into `delete-one` and `delete-all`. I.e. Splitting the delete feature into the 2 abilities: to delete 1 specific show, and to delete all shows of the specified show name.
  This design would use 2 separate commands, `DeleteOneCommand` and `DeleteAllCommand`. The user would use "`delete-one SHOW_NAME|SHOW_DATE`", and "`delete SHOW_NAME`" as the input format for the function to delete one show, and to delete all the specified shows respectively.

  - Pros: `delete-all` would enable the user to delete all the shows of specified show name with less hassle, without having to input all the dates.
  - Cons: Having 2 delete functions is less intuitive and may confuse new users, making them more prone to deleting multiple shows unintentionally.

- **Alternative 2 (Current Implementation):** One common delete feature that can delete multiple shows at once.
  - Pros: This method is intuitive and flexible, as it enables users to delete multiple shows with a single command. This makes deletions faster, and users will save time by typing less. This is important since this is a command-line based application.
  - Cons: If the user wishes to delete all shows of a specific name, the user must input all the dates which the show is scheduled for. This can in turn slow down the user instead. However, the instances of this happening are less likely.

Excerpt 2: Documentation on the Delete Command from Optix Developer Guide

As can be seen from Section 4.3.1, the key operations utilized in the `Delete` Command were listed, along with the classes which these commands belong to. This section is intended to serve as a glossary for developers who wish to know the workings of this command or want to modify this feature. This is important as some of these operations are defined in other classes(such as the `Model` class), and its definition cannot be found in the DeleteCommand class. Hence developers who refer to the glossary could use it to easily and quickly find where the operations are implemented.

Additionally, class diagrams and state diagrams were used extensively throughout the developer guide. These diagrams are a form of Unified Modelling Language(UML) diagram used in the field of software engineering to visualize a system. Like the user guide, the purpose of these diagrams is to serve as visual guides for developers. These diagrams provide transparency in the class/ system architecture in ways that text cannot. Developers who are trained to interpret UML can understand the class architecture at a glance, without having to sort through many lines of code. Developers who intend to modify the feature, or software engineers that need to appraise the code will require less time reading and understanding the code. This will aid developers in creating applications and extensions based on this software.

Code snippets were provided to highlight key logic of the Delete command. This is primarily targeted at developers who intend to implement the delete function on their own applications. The code adheres to the Java Coding Standard for Components[2], and developers would quickly be able to understand the code just by reading it. Additionally, by displaying the code snippet developers can quickly get started on their own implementation of the `delete` function by simply copying and pasting from the document. This will surely save time when compared to finding the same snippet from the code base.

To facilitate the design process for potential software developers who wish to improve this software, our(the developer team) design considerations were provided. Alternatives to the current implementation was given, and their respective pros and cons were listed. Developers would be able to understand the direction that we wanted to take when building this software, and those that feel our current implementation is inadequate can read the alternative implementations to get suggestions for new implementation designs.

Conclusion

This portfolio provides a brief overview of my contributions to the code, User Guide and Developer Guide of the Optix software. Steps were taken to justify my design choices for the guides and code. Most of my contributions aimed to provide readable and adaptable code, as well as detailed documentation that is easily understood by the laymen(User Guide) and software developers(Developer Guide). A main takeaway is that conscious effort must be made to ensure there is a high level of transparency behind feature design. This is useful for both developers and users. Developers would be able to better adapt the code for their own use. While it is not a requirement, users who understand the logic behind the program would also be able to swiftly learn how to use it.

[2] **Coding Standards for Components:** It is recommended to write components name by its purpose. This approach improves the readability and maintainability of code.