

Liu Wei Jie Nicholas — Project Portfolio for Optix

About the project

My team is a group of 4 Computer Engineering Students. My team and I were tasked to enhance a basic command line interface(CLI) desktop Duke application while retaining the CLI feature for our Software Engineering Project. We agreed on morphing it into a seat booking cum finance managing system for a venue such as a theatre. Our product, Optix, allows users like theatre managers to track seat booking of a show as well as the finances from the bookings.

A snapshot of our product is as shown:

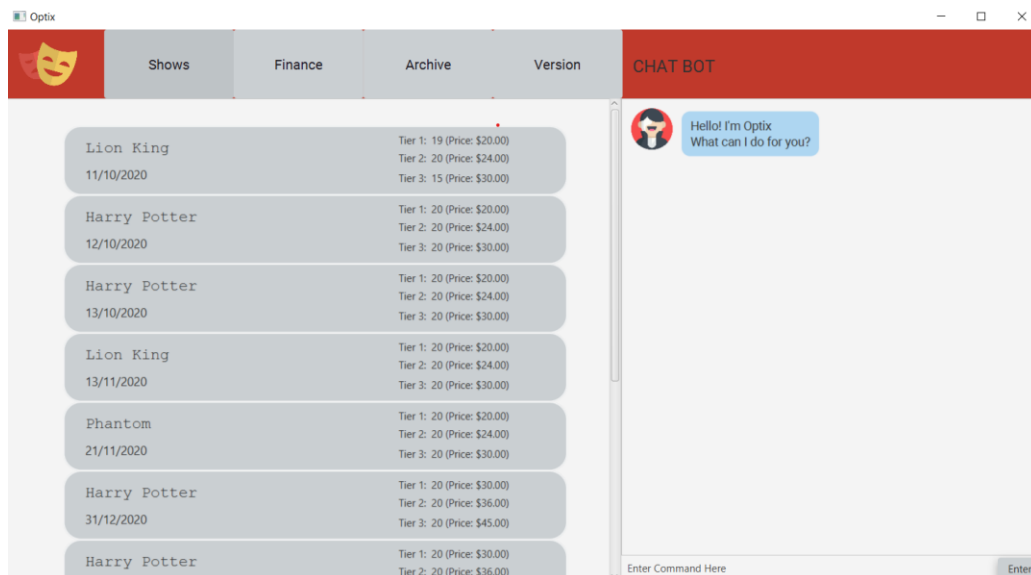


Figure 1: The Graphical User Interface for Optix

Grey highlighted components indicate that this is a command or feature.

My role was to implement the code for the commands dealing with the finance features as well as maintaining the documentation of the code. The following sections elaborate on my enhancements as well as on my contributions to the user and developer guide.

Summary of contributions

This section summarizes my coding, documentation and other contributions made to the team's progress.

Enhancements added

1. I implemented the finance features of Optix, which allowed users to view the profits earned from a single show or from a particular month.
 - a. What it does: The `view-profit` command allows the user to view the profit earned from selling the seats of one show. The `view-monthly` command allows the user to view monthly profits.
 - b. Justification: Users like theatre managers would want to view the profits earned from the shows without having to scroll through the entire list. Users also do not have to go through the trouble of manually finding shows and calculating the monthly earnings with the `view-monthly` feature.
 - c. Highlights: This enhancement can act as a baseline for other commands such as viewing profits for multiple shows or dates at once. The implementation was also challenging as the show lists are split into the current list(future shows) and archive list(past shows) so appropriate data structures need to be used to retrieve the shows from their corresponding list.
2. I also implemented the command to `re-assign` the seats of customers.
 - a. What it does: The `re-assign` command allows the user to change the seat booking of a customer to another seat.
 - b. Justification: Customers may have booked the wrong seat or mistakes might be made by the managing personnel when booking the seats. Customers might also want to change their seats based on the distance from the stage.
 - c. Highlights: Other features such as removing or refunding seats would require certain methods implemented by `re-assign` command. The `re-assign` feature also calculates the cost difference between the changed seats and requires the finances to be updated accordingly.

Code contributed: Click the link below to see a sample of my code: [Functional code](#)

Other contributions:

- Project management:
 - There were a total of 4 releases, from version mid v1.1 to v1.4 as of current date. I managed releases versions 1.2 and 1.3 (2 releases) on GitHub.
 - I was also in charge of ensuring that weekly increments were delivered on time.
 - Wrote test cases to increase test coverage.
- Documentation:
 - Implemented the layout of the User Guide and Developer Guide on github.
 - Did the manual testing in the Developer Guide. Pull request [#146](#)
 - Made cosmetic improvements to the User Guide to make it more user-friendly.

- Community:
 - Reviewed Pull Requests: [#12](#), [#13](#), [#21](#), [#68](#), [#69](#), [#78](#), [#102](#), [#111](#), [#115](#), [#116](#)
 - Offered suggestions to team members to refactor some parts of the code for tidier implementation.

Contributions to the User Guide

I updated the User Guide whenever a new feature is added. I also ensured the User Guide is readable and user friendly. The sections below show the contributions I made to the User Guide.

The figures below show some of the contributions I made to the User Guide.

3.5. Finance Commands

The following set of commands helps with tracking the finance of the theatre.

3.5.1. View the profit of a show: `view-profit`

Displays the profit earned from that particular show. Format: `view-profit SHOW_NAME|SHOW_DATE`

- Displays the profit for the specified `SHOW_NAME` on `SHOW_DATE`
- Displays projected earnings for a show if `SHOW_DATE` is in the future.

Example:

```
view-profit Lion King|5/5/2020
```

3.5.2. View the amount earned for a particular month: `view-monthly`

Displays the profit earned for a particular month.


 use numbers or abbreviations to represent the month instead of spelling it out!

Figure 2: UG Finance feature pt1

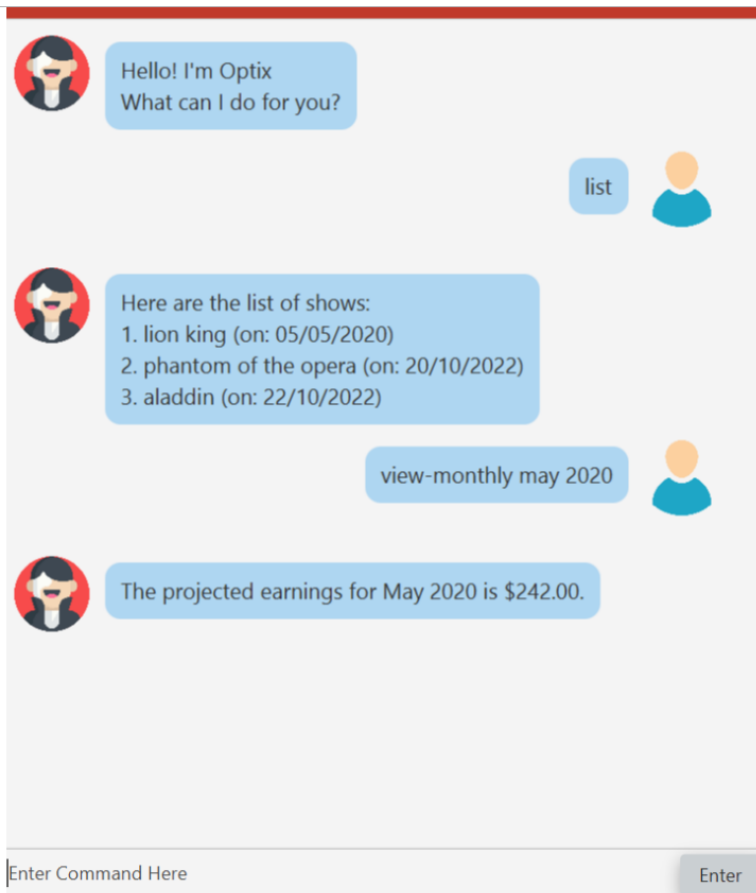
Format: `view-monthly MONTH YEAR`

- Displays the total profit collected for all the shows in MONTH YEAR
- Displays projected earnings if MONTH YEAR is in the future.

Examples:

```
view-monthly 1 2020
view-monthly Jan 2020
view-monthly January 2018
```

Figure 3: UG Finance feature pt2



As shown in the figure above, entering the `view-monthly` command would cause Optix to respond with the profit of that month.

Figure 4: UG Finance feature pt3

As shown in the figure above, entering the `view-monthly` command would cause Optix to respond with the profit of that month.

	Show	Finance	Archive	Version
	lion king 05/05/2020	Revenue: \$242.00		
	phantom of the opera 20/10/2022	Revenue: \$0.00		
	aladdin 22/10/2022	Revenue: \$0.00		

As shown in the figure above, the GUI would also display the shows performed in that month, together with the revenue of each show

Figure 5: UG Finance feature pt4

I provided examples of the usage of commands to make the User Guide more user friendly as shown in Figures 2 and 3. I also included a tip to inform users of shortcuts which can be seen in Figure 2.

I also included images of the GUI so that users know the expected outcome when implementing the features as seen in Figures 4 and 5.

Contributions to the Developer Guide

I updated the Developer Guide to include the new features and to make it more user friendly. I also wrote the instructions for manual testing.

Finance features of the Developer Guide

The following figures contain snippets of the developer guide which explain my implementation of the finance features.

3.2. [Proposed] View Monthly Revenue Feature

3.2.1. Proposed Implementation

Viewing the monthly revenue is executed by the `ViewMonthlyCommand`, which extends from the abstract class `Command` and is stored under the `Command` package.

Additionally, it implements the following operations based on the query date:

- * `Model#findMonthly()` — Retrieves the list of shows in the month specified by the input.
- * `Model#getShows()` — Retrieves the current list of shows should the user query be in the future.
- * `Model#getShowsHistory()` — Retrieves the archive list should the user query be in the past.
- * `OptixDateFormatter#getMonth()` — get the integer value of the month.
- * `OptixDateFormatter#getYear()` — get the integer value of the year.
- * `Theatre#getProfit()` — get the profit earned for the show.

Given below is an example usage scenario of the `ViewMonthlyCommand` at each step.

Step 1

The user starts the application. `Storage` will be initialised with the saved contents from previous runs. `Model` will then be initialised and the current list and archived list of shows are loaded into `Model`.

Step 2

The user executes `view-monthly June 2017` to check the revenue earned by all shows in June 2017. Once `Parser` verifies that the command is of correct format, `ViewMonthlyCommand` calls `OptixDateFormatter#getMonth()` and `OptixDateFormatter#getYear()` to get the integer values of month and year respectively.

Step 3

`ViewMonthlyCommand` calls `Model#getShowsHistory()` since the date is in the past. This hashmap of `ShowsHistory` is then passed into the `Model#findMonthly()` of the `Model` where a list of the shows in the specified month is created.

Step 4

The profit for each of the shows in the remaining list is then added up in `Model` by calling `Theatre#getProfit()`.

Figure 6: DG Finance feature pt1

The following sequence diagram shows how the view-monthly operation works:

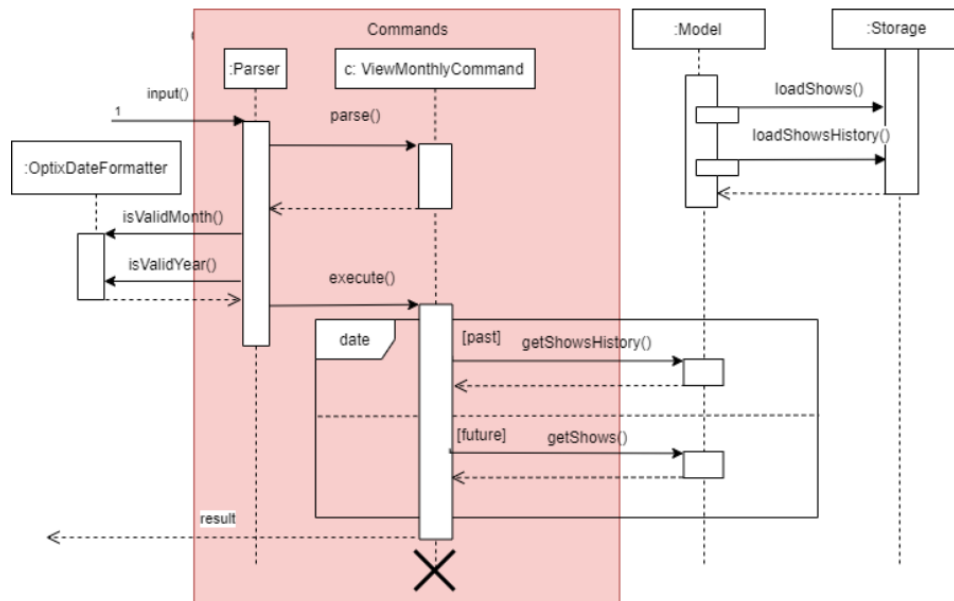


Figure 7: DG Finance feature pt2

The following activity diagram summarizes what happens when a user executes the ViewMonthly Command:

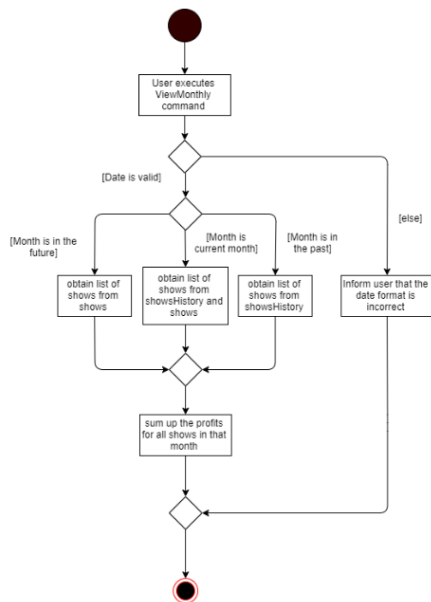


Figure 8: DG Finance feature pt3

4.1.2. Design Considerations

Aspect: How view-monthly executes

- Alternative 1 (current choice): Obtain the required shows from the current and archive list separately.
 - Pros: Reduces search time if Optix knows which list to search from.
Also increases user friendliness since message can be customised depending on whether the show is from the current or archive or both lists.
 - Cons: More bug prone since there are more conditional statements.
- Alternative 2: Combine the current and archive list before searching for the shows.
 - Pros: More efficient since the entire list would be sorted.
 - Cons: Message showing the monthly revenue will always be the same, making it less user-friendly.

Figure 9: DG Finance feature pt4



I listed the key methods used by the `view-monthly` command as seen in Figure 6 which would allow developers to know how the classes are connected. In addition, developers can add on to the code if they know the `view-monthly` class is missing certain methods. I also drafted the sequence and activity diagrams as shown in Figures 7 and 8 respectively. By combining text and visual aids, developers would be able to grasp the implementation more easily. The design considerations I included as shown in Figure 9 would allow other developers understand my thought process and why I used certain methods over others, which would aid them in future enhancements accustomed to Optix.

Testing instructions

The figures below show snippets of the testing instructions.

Appendix G: Instructions for manual testing

The instructions to test the program manually are given below.

-  These instructions are meant to provide a starting point for testers to work on. Testers are expected to conduct more *exploratory* testing.
-  All commands, show names and parameters are case insensitive.

G.1. Launch and Shutdown

1. Initial launch

- Download the latest `optix.jar` [here](#) and copy it into an empty folder.
- Double-click the jar file
Expected: Shows the GUI with the display window on the left(empty) and Chat Box on the right.

Figure 10: Test instructions pt1

G.9. Viewing a show

1. Display the layout of a show with `SHOW_NAME` on `SHOW_DATE`
 - i. Prerequisites:
List all shows using the `list` command. There should be multiple shows in the list.
There should be a show with the specified `SHOW_NAME` on `SHOW_DATE` in the list.
 - ii. Test case: `view aladdin|5/5/2020`
Expected: Display window shows the seat layout of aladdin on 5/5/2020. The red seats indicate they are booked while the blue seats indicate they are available. Chat box displays a message showing the layout of the show with a ✓ indicating the seats are booked and a ✗ indicating the seats are available.
2. Non-existent show
 - i. Test case: `view non-existent show|11/5/2020`
Expected: Display window does not change. Chat box displays a message showing that the show cannot be found.

G.10. Re-assigning seats of a show

1. Changes the `SEAT` of a customer for a show with `SHOW_NAME` and `SHOW_DATE`
 - i. Prerequisites:
List all shows using the `list` command. There should be multiple shows in the list.
There should be a show with the specified `SHOW_NAME` on `SHOW_DATE` in the list.
 - ii. Test case: `reassign-seat aladdin|5/5/2020|A1|F8`

Figure 11: Test instructions pt1

I included several test cases which testers can copy and paste into the chat box of Optix when they are conducting the trials. I also included prerequisites needed for the test cases so that the testers know exactly how the features work.

Conclusion

This product portfolio summarises the contributions I have made to the project. The contributions I made in the code were mostly finance related, which is one of the main selling points of Optix. I also tried to ensure the User Guide and Developer Guide were easy to navigate and understand so that users do not have to refer to these guides again after going through minimal number of times. It is my goal to further enhance the code or any of the guides as much as possible to avoid time wastage.