Rusdi Haizim – Project Portfolio for JavaCake

About the project

My team and I were tasked with enhancing a basic command line interface desktop application that serves as a personal assistant for our Software Engineering project. After much deliberation, we decided to shape it into a Java learning assistant application called ***JavaCake***, which allows you to interact with a Command Line Interface (**CLI**) while still viewing a Graphical User Interface (**GUI**).

There is not only learning material to learn Java, but also interactive quizzes and other features. Through this application, prospective students to the CS2113/T course as well as others curious to learn Java can easily learn the fundamentals of the Java programming language without needing to cross-reference with heavily-documented and daunting websites.

This portfolio serves to document my contributions to the project and the role that I played in developing it.

My role was to design and write the codes for the Profile, deadlines and **GUI** features. The following sections illustrate these enhancements in further detail. I have also included the relevant sections that I have contributed to the User and Developer guides with respect to these enhancements.

Note the following symbols and formatting used in the following sections for easier comprehension:

ℹ	This symbol indicates important information.
💡	This symbol indicates a tip that you can follow in order to achieve the best result from the given information.
`list`	A grey highlight (called a mark-up) indicates that this is a command that can be inputted into the command line and executed by the application.
`Logic`	Blue text with grey highlight indicates a component, class or object in the architecture of the application.
`isQuiz`	Italicised Consolas font will be used to denote attribute names inside a Java class.

Summary of contributions

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

Enhancement added:

- Addition of a Profile feature.
  - What it does:
    This feature allows your progress to be tracked and continuously updated throughout the app’s runtime.

    - Justification:
      This is needed since whenever you exit the app and relaunches it, their progress should have been previously saved and restored.
    - Highlights:
      This enhancement serves as the skeleton to other components my groupmates had implemented such as the quiz questions implemented through `Model`.
- Addition of the ability to add deadlines.
    - What it does:
      The `deadline` command allows you to add a deadline for them to pace their progress when using the app. You may also set the deadline as `done`, `snooze` or `delete` their deadlines.
    - Justification:
      This feature works as a progress tracker, where you can add whatever content to get done, and by what date.
    - Highlights:
      This enhancement adds a more complete user experience as when you first launch the app, you will be greeted with a list of the deadlines that had been previously set.
- Addition of the **GUI**.
    - What it does:
      The **GUI** allows you to view the current content or go through the quiz while multitasking other tasks, such as creating notes as well as adding deadlines.
    - Justification:
      With a **GUI**, you can still utilise the app's main features while being aware of your current deadlines/notes, compared to a single-view mode utilised by most **CLI** apps.
    - Highlights:
      You can now multitask when using different functions simultaneously, which enhances the user experience. There are also some aesthetically pleasing features like animated pictures and scrolling text that adds to the experience.

Code contributed:

Please click these link to see a sample of my code: [Functional code]

Other contributions:

- Project management:
    - There were a total of 4 releases, from version 1.1 to 1.4. I managed releases versions 1.1 and 1.2 on GitHub.
- Enhancements to existing features:
    - Updated the **GUI** to support dual colour schemes (Light and Dark mode) to allow varied user experience. (Pull request #91)
    - Updated the **GUI** to include animations for
- Documentation:
    - Converted the Developer Guide from GoogleDocs into AsciiDoc and uploaded onto GitHub for online perusal.
- Community:
    - Reviewed Pull Requests (with non-trivial review comments): #74, #101, #121
- Tools:
    - Integrated a third party library (Commons IO) to the project (#106)
    - Integrated a new GitHub plugin (Travis) to the team repo (#14)

Contributions to the User Guide

JavaCake contains a myriad of commands that you can execute. These are some of the common commands that you may execute:

`list`	Lists down the topics/quizzes that you can view/do in the dialog box.
`goto`	Displays the content/directory as specified by the argument that follows the command.
`exit`	Displays an exit message before closing the application.
`deadline`	Adds a new deadline with a specified date.

The following section is an excerpt from our ***JavaCake*** User Guide, showing the additions I have made for the deadline feature.

Adding a deadline: `deadline`

This command allows you to add a task which is due by the date specified.

What's the purpose?

This serves as a form of progress-tracker, where you can set what topics you want to be done or what quizzes you want to finish and by what time.

Example:

Let's say that you want to set a deadline to complete the topic 'Java Basics' by the 11th of November.

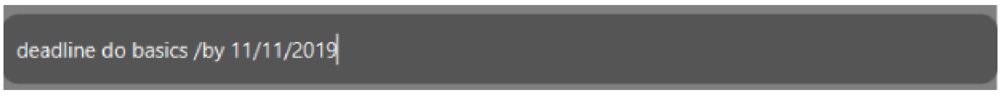
The `deadline` feature can only accept certain date formats.

Some common acceptable formats include the following examples:

- 01/01/2019
- 1 january 2019
- 01 01 19 2359

Refer to Developer Guide's Section 3.7 for comprehensive list of acceptable date formats

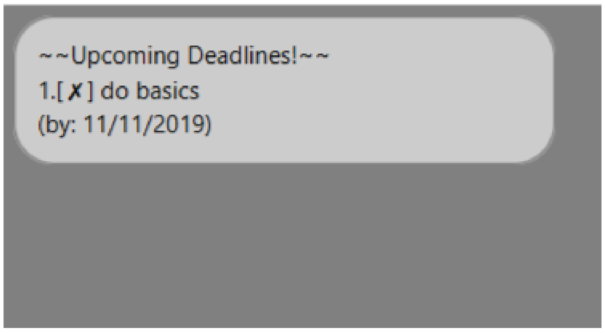
1. Type `deadline do basics /by 11/11/2019` into the command box and press the 'Enter' key inside the command box.



```
deadline do basics /by 11/11/2019
```

Figure 1: Input deadline in the command box

2. The newly added deadline will be shown in the following format below, near the top-right side of the app along with the list of deadlines that were previously entered. They are sorted in the order of most recent dates.



```
~~Upcoming Deadlines!~~
1.[✗] do basics
(by: 11/11/2019)
```

Figure 2: List of deadlines at top-right side of the app

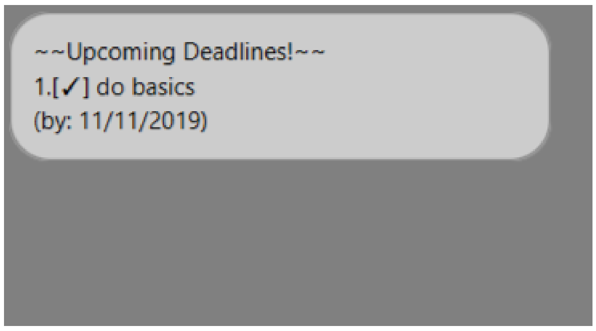
Meeting a deadline: `done`

This command allows you to set a deadline as done.

Example:

Let's say that you have just finished browsing through the 'Java Basics' content and want to set your previous deadline as done.

1. Following from the previous input, type `done 1` into the command box and press the 'Enter' key or 'Send' button on the right of the command box.
2. Your list of deadlines will now be updated, with the selected deadline having its 'X' symbol replaced with a '✓' instead, indicating that the deadline has been met.



```
~~Upcoming Deadlines!~~
1.[✓] do basics
(by: 11/11/2019)
```

Figure 3: Selected deadline is labelled as 'done'

Refer to Sections 3.9 and 3.10 of the User Guide for similar instructions on how to use the `delete` and `snooze` commands respectively.

Contributions to the Developer Guide

The following section shows my additions to the ***JavaCake*** Developer Guide for the Profile and **GUI** features.

Profile feature

This is a backend feature that keeps track of your details such as username, individual quiz scores and overall quiz scores. It is used in conjunction with `Model` objects like `QuizSession` and `ReviewSession` that my other groupmates have implemented. The following shows some of the common operations:

`profile#getUsername()`	Gets the username.
`profile#resetProfile()`	Resets the current profile, along with their respective data after calling the `reset` command.
`profile#setIndividualMarks()`	Sets the marks for a particular quiz in a particular difficulty (specified in the function parameters).
`profile#writeProgress()`	A private method which overwrites the current quiz scores inside the save file with newly updated scores.
`profile#getTotalProgress()`	Gets the overall marks for all the quizzes.

These operations are exposed in the `StorageManager` a class whose object is passed as a parameter when executing various other commands from `Logic`.

Given below is an example usage scenario of how `Profile` behaves when the quiz is finished.

1. You have started a `QuizSession` for Quiz 1 on Easy difficulty and have obtained 3 out of 5 marks. (Refer to Developer Guide to learn more about how `QuizSession` works)
2. The `profile#setIndividualMarks()` is called which stores the value of the marks obtained for that particular quiz type and difficulty.
3. The `profile#writeProgress()` is called within the Profile, which overwrites your current quiz scores inside savefile.txt.

GUI feature

This is a frontend feature which mainly serves to allow you to multitask while using the application. Tasks such as viewing content/doing the quiz, creating deadlines and creating notes are designed to be mostly mutually exclusive, so as to allow you to focus more on the content, which is the ***JavaCake's*** main feature.

Currently, the **GUI** has a total of 5 viewable interfaces, along with an additional *textfield* to input commands.

View	**Description**
`TopBar`	This view contains the title of the app, ***JavaCake***, as well as the individual and total quiz scores at the side.
ContentDialogBox	This view displays the main content and quizzes, which mainly changes according to commands like `list` and `goto`.

DeadlineDialogBox	This view displays the list of deadlines, which is automatically updated (in sorted most recent order) after every deadline-related command or after the `reset` command is called.
NoteDialogBox	This view displays the list of notes, which is automatically updated after every note-related command or after the `reset` command is called.
`AvatarScreen`	This view displays our app's avatar. The avatar changes its animation to blink every few seconds and whenever different quiz scores are obtained. There is also an additional dialog box below which displays some motivational quotes.

`TopBar` and `AvatarScreen` are highlighted as such since they themselves are Java classes, as compared to the other views which are implemented using the `DialogBox` class.

The diagram below depicts a simplified activity diagram on the current implementation of user input processing in the **GUI**, since the other features are similar in nature and only differ in the type of Boolean flags required.

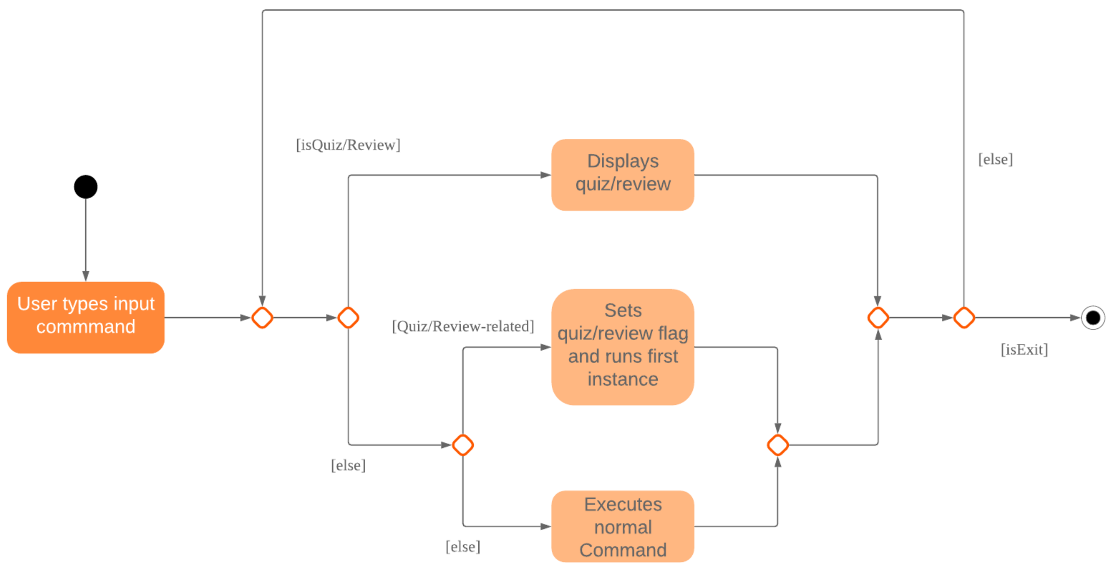


Figure 4: Simplified Activity Diagram for user input processing (for quiz/review only)

Given below is an example usage scenario of how `MainWindow` , the class that handles **GUI** inputs, behaves when a command is inputted.

1. User types in command `list` followed by `goto 4.1`.
2. `JavaCake` handles the parsing of the inputs and realises that the command is quiz-related. (Refer to Developer Guide for more information about the classes mentioned)
3. The *`isQuiz`* flag is set to true and the first quiz question is shown in the **GUI**.
4. While you continue answering the quiz questions, `MainWindow` will run the corresponding inputs through the *`isQuiz`* block, which continues displaying the following quiz questions.
5. Upon the last quiz question, the *`isQuiz`* flag is set to false and the *`isReview`* flag is set to true, giving you the choice to execute a `ReviewSession` or go back to normal operation by typing `back`.

Design Considerations

When designing the `MainWindow#handleUserInput`, I had to make a decision on how best to parse and handle the inputs which causes ***JavaCake*** to go into quiz, review and other modes which block the usual input operation. I also had to deliberate on how to display the quiz scores inside `TopBar`. The following is a brief summary of my analysis and decision.

Aspect	**Alternative 1**	**Alternative 2**
User input for Quiz and Review	Implement a *textfield* swap and new methods just to handle the `QuizSession`, `ReviewSession`, and other commands that block the initial textfield. **Pros:** Easy to implement **Cons:** Will be more memory-intensive and not scalable.	Keep track of different modes of operation via Boolean variables like *isQuiz*, *isReset* and such. **Pros:** Easy to implement **Cons:** Normal commands will usually not be executed when these Boolean variables are set. **I decided to proceed with this option** because it has a lesser burden on the CPU. The execution of normal commands can also be resolved when the order of if-else statements are implemented properly.
Display of Quiz Scores in TopBar	Dynamically create progress bars and progress wheel objects according to the number of Quiz text files found in the resources folder. **Pros:** Scalable for future implementations of additional quizzes **Cons:** Hard to implement and prone to errors if quiz files are not found or properly formatted.	Create a fixed number of progress bars and progress wheel objects to display. **Pros:** Easy to implement **Cons:** Not scalable, since if another developer wants to add another quiz, they have to modify the source code directly. **I decided to proceed with this option** as for this project's scope, limiting the number of quizzes to be only 4 makes it easier for us to test and debug our **GUI**.