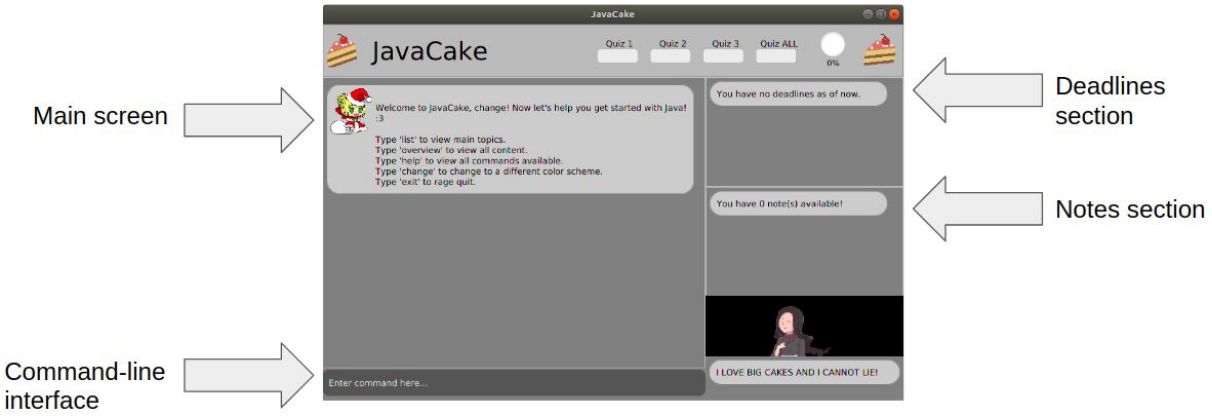
# Kishore R – Project Portfolio for JavaCake

## Introduction

The purpose of this portfolio is to showcase my technical abilities in a project called JavaCake. My team and I were tasked with enhancing a basic command-line interface desktop application that serves as a personal assistant. We decided to shape it into a Java learning assistant application called JavaCake, which incorporates not only the learning material to learn Java but also contains interactive quizzes, progress tracking as well as note-taking. Through this application, prospective students to the CS2113/T course and others interested in learning Java can easily learn the fundamentals of the Java programming language without needing to cross-reference with heavily-documented and daunting websites.

My role was to expand the quiz feature, enhance the goto function and code the helper feature. The following sections illustrate these enhancements in further detail. I also included the relevant sections that I have contributed to the user and developer guides with respect to the aforementioned enhancements.

The figure below is an illustration of our project:




Feature 1 : GUI of JavaCake

The GUI contains the Main screen, in which most of the interaction will occur, a command-line interface that users can type in their commands to use JavaCake. Users can also set deadlines and notes which will appear in their sections respectively.

The following symbols and formatting will be used in this document:

This symbol indicates important 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.

*currentFilePath* Italicized Consolas font will be used to denote variable names.

## Summary of contributions

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

### Enhancement added:

- Added the three difficulty levels for the quiz.
  - What it does: It allows the user to learn incrementally.
  - Justification: This gives the user more detailed feedback on what their level of understanding is for a certain topic.
  - Highlights: To achieve this functionality, changes had to be made to the other class like profile to store a larger number of scores for the increased number of quizzes.
- Enhanced the ability for users to navigate into a file or directory directly using the `goto` command.
  - What it does: Allows users to type in 1 goto command instead of multiple goto commands.
  - Justification: This increases the user's navigability, and returning users who know the file path can easily access JavaCake's materials.
  - Highlights: This feature works in hand with the overview command, where if a user wishes to view a file listed form the overview command, the user can do so by just appending the index to the goto command.
- Added a helper function.
  - What it does: Assists the user in prompting the correct type of command when the command has a typo.

- Justification: The number of commands available in JavaCaka might be overwhelming for a new user to get used to, and the helper command can assist the user in using JavaCake.
- Highlights: This enhancement works can easily be expanded to include future commands.

## Code contributed:

Please here to see a sample of my code.

## Other contributions:

- Project management:
  - Opened #48 to #58 of issues of use cases.
- Enhancements to existing features:
  - Refactored quiz to increase scalability (Pull request #97, #98)
  - Fixed bugs that were apparent when running in Linux operating system. (Pull request #68, #107)
  - Wrote additional tests for goto and quiz to increase quality assurance for the functionality of the features. (Pull request #162, #168, #172)
- Documentation:
  - Added figures in the User Guide and Developer Guide to make it more reader-friendly.
- Community:
  - Reviewed Pull Requests.
  - Reported bugs for other teams in the class. (Pull Request #171, #175)

# Contributions to the User Guide

We had to update the original User Guide with instructions for the enhancements that we had added. The following is an excerpt from our JavaCake User Guide, showing the additions I have made for the quiz and helper features.

## Doing a quiz

*JavaCake* offers quizzes of varying difficulties for every topic for you to reinforce your understanding of the topics incrementally.

To start a quiz,

1. Type goto to navigate to any subtopic with the name “Test Yourself!”

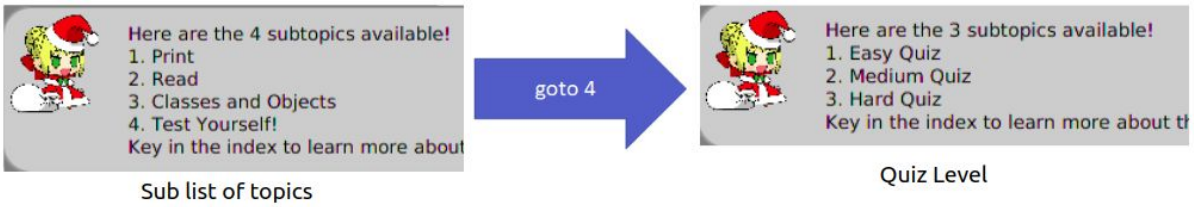



*Figure 2: Quiz level displayed*

2. Type goto to choose your difficulty.

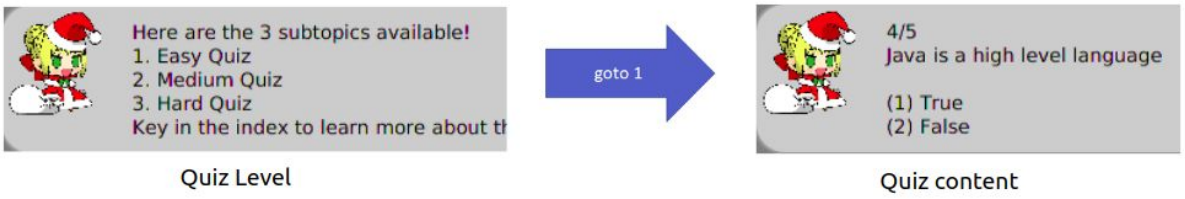



*Figure 3: Quiz launched in the dialogue box*

You can proceed to answer the multiple choice questions. Once you have answered all the questions in the quiz, *JavaCake* will show you a results screen that shows you how many questions you answered correctly, as shown in the figure below.

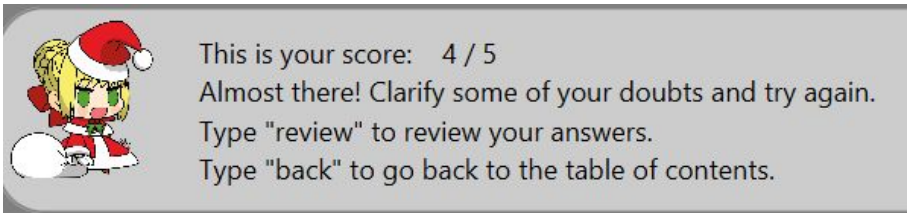



*Figure 4: Results screen*

Additionally, if your new score is higher than your previous score for that topic and difficulty, your total progress will increase, reflected in the progress bars in the top bar. In order to get complete JavaCake, you need to score full marks for all the quizzes in all three difficulties. There is an unlimited number of attempts for any quiz in JavaCake, and JavaCake will only remember your maximum score for each quiz. If you wish to start from scratch as a new user, you can delete your progress by typing the reset command. (Refer to the user guide section 3.18)

From the results screen, you may choose to review your answers by typing review or return to the table of contents by typing back as shown in Figure 3. (Refer to user guide section 3.6)

**All other commands (EXCEPT exit) are disabled during the quiz. JavaCake will only accept valid integer inputs that correspond to the available options for that question.**

You cannot exit the quiz until you have completed all the questions in that quiz and exited the results screen of the quiz.

**Do not close the application in the middle of the quiz.**

If you close the application during the quiz session, progress for that specific quiz session will be lost and you would have to re-do the session.

## Helper for commands:

If you have made a small typo in inputting a command, JavaCake will help you out by replying to the correct command.

For example, if lisr is entered instead of list as shown in *figure 5* below, JavaCake will prompt you for the correct command, as shown in *figure 6*.


*Figure 5: User inputting incorrect command*

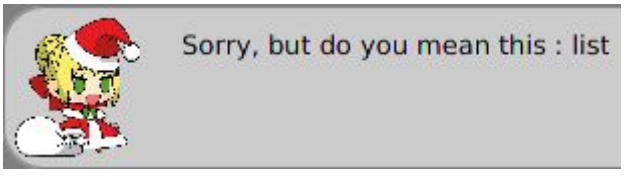



*Figure 6: JavaCake helping user to use the correct command*

To a new user, these commands may be slightly hard to get used to at first, but with this helper function,  the user get used to the overall flow of the application.

## Contributions to the Developer Guide

The following sections show my additions to the JavaCake Developer Guide for the quiz feature.

### Quiz feature

#### Proposed Implementation

This quiz feature allows users to test themselves and review the questions if they wish to do so. The figure below is an activity diagram to show how quiz and review features will interact with the user.

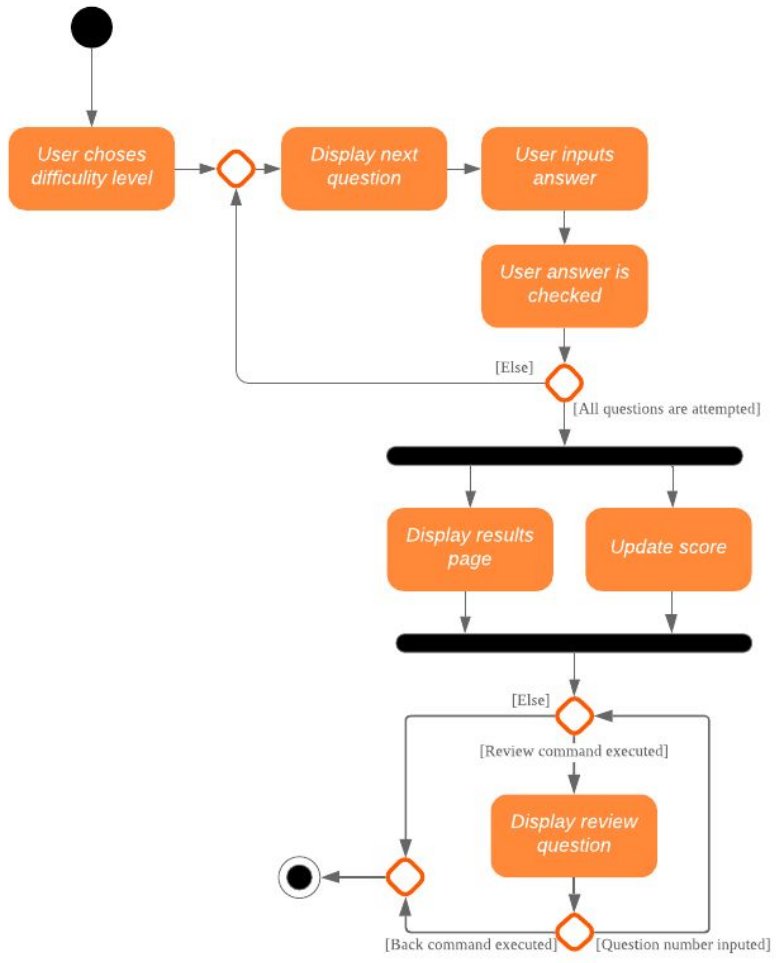



Figure 7: Activity diagram for Quiz and and Review features

In accordance with the diagram as shown above, here is an example of JavaCake interacts with the user attempting to do a quiz.

1. Quiz session starts with the user choosing the difficulty level.
2. The question type *qnType* and level of difficulty *qnDifficulty* are both set by reading the *filePath* from `Logic`.
3. A `QuestionList` object, containing an ArrayList of `Question` objects, is initialized by reading all text files containing the questions in the current directory.
4. *MAX_QUESTIONS* denotes the number of questions to be selected from the ArrayList of `Question`. `QuestionList` selects *MAX_QUESTIONS* number of questions randomly .
5. `QuizSession` will use this `QuestionList` object to get the list of questions to be displayed to the user.
6. User inputs his/her answer. *currScore* represents the score of a user for a single quiz session. Therefore, for every correct answer, *currScore* will increase by 1.
7. When the user answers all the questions, a results screen will be shown to the user, displaying the final *currScore* out of *MAX_QUESTIONS*.
8. Custom messages and different avatar expression will be displayed as well, determined by a *scoreGrade* of `BAD`, `OKAY` or `GOOD`, which is dependent on the score for that particular quiz session.
9. If *currScore* is higher than the user's previous score that was stored in `Profile`, the new score will be updated. `TopBar` will also be updated to show a visual change in the progress bar.
10. If user wants to review the questions,he/she can type the `review` command.
11. This will initialize `ReviewSession`, where the questions from the previously completed quiz sessions are shown again along with the user's answer and the correct answer.
12. User can input integers that correspond to the question number in order to jump to that question. eg. Typing '3' to review questions number 3.
13. After user is satisfied with reviewing the questions, the user inputs the `exit` or `back` command
14. `ReviewSession` will terminate as shown in the diagram above.

Both `QuizSession` and `ReviewSession` implement the interface `QuizManager`.

The important methods for the quiz feature are as follows:

1. `QuestionList#loadQuestions()` — loads selected questions in text files and stores them in an ArrayList.
2. `QuizSession#getQuestion()` — returns the question string.

3. `QuizSession#getQuizResult()` — returns the text of the result screen, including *currScore*.
4. `QuizSession#overwriteOldScore()` — updates both the score in `Profile` to the new score from the quiz session and the `TopBar` if the new score is higher than the score in `Profile`.

Please refer to the Developer Guide for more information on `TopBar` and `Profile`.

## Design Consideration

There were several considerations that were made when designing how the quiz feature interacts with storage. Table 1 below summarises the alternatives and choices made for the implementation of loading questions from storage.

*Table 1: Design considerations for Quiz implementation*

| Aspect | Alternative 1 | Alternative 2 |
|---|---|---|
| Quiz content storage method | `QuestionList` class contains and maintains all hardcoded Question objects and the number of quizzes that each topic contains.<br><br>**Pros:** Easy to implement and test as it is not susceptible to IO or File exceptions that may arise from reading from an external file.<br><br>**Cons:** As all questions and answers have to be hardcoded within the class, it is not scalable as number of quiz questions increases. | Quiz questions are stored with their answers in the text files. `QuestionList` is to load the relevant questions from these text files.<br><br>**Pros**: Very scalable as additional questions and answers can be easily added without having to manually change the code.<br><br>**Cons:** Each text file that contains the quiz must follow a certain naming format.<br><br>**(Current choice)**<br>This method makes it easier to expand quiz content just by adding files without changing the code. |