

# Claire Chan Yen Hwa— Project Portfolio for JavaCake

## About the project

My team was given the task of enhancing a basic command line interface desktop application that serves as a personal assistant for our CS2113T Software Engineering project. We decided to shape it into a Java learning assistant application called *JavaCake*. This application facilitates users' learning of basic Java using easy-to-understand tutorials and interactive quizzes, with additional features such as progress tracking and note-taking. The target audiences for this application are prospective students to the CS2113/T course as well as inexperienced programmers wanting to learn Java, so that they can easily learn the fundamentals of the Java programming language on a centralized platform.

My role was to design and write the code for the Quiz and Review feature. The purpose of this document is to detail my contributions to this project, such as the function and implementation of the Quiz feature and sections relating to the Quiz feature that I have written in the user and developer guides.

The figure below shows the landing page of our application *JavaCake*.

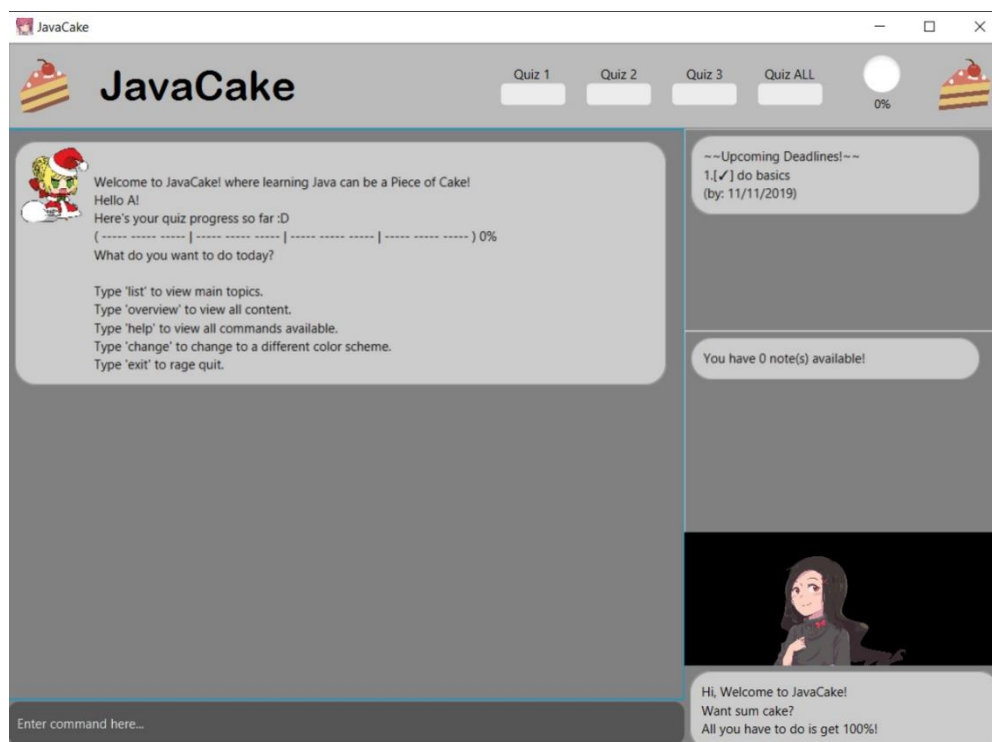



Figure 1: *JavaCake* landing page

Note the following symbols and formatting used in this document:

	This symbol indicates important information.
<code>list</code>	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.
<code>Logic</code>	Blue text with grey highlight indicates a component, class or object in the architecture of the application.
<i>currentFilePath</i>	Italicised 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

- ❖ I added a feature for users to answer quizzes.
  - What it does: When the user initiates a quiz, *JavaCake* randomly selects a few multiple-choice questions from a bank of questions stored in the application, shows the selected questions to the user and checks the user's answers to the questions. When the quiz is completed, the application tabulates the user's results.
  - Justification: Simply reading tutorials are not enough for users to absorb the learning material. An interactive quiz presents users with the opportunity to reinforce their learning and correct misconceptions early.
  - Highlights: The difficulty in implementation of this feature comes from processing user input. As the input stream is shared with the main program, there is a need to block out all other commands during a quiz so that users cannot navigate away from the quiz until it is completed. This means that separate program states and parsers must be implemented in order to minimize bugs.
  - Credits: The idea to integrate a quiz feature into *JavaCake* was inspired by the website [VisuAlgo](#). VisuAlgo, a website that teaches and visualizes data structures and algorithms, has a training feature for users to answer semi-randomly generated quizzes for selected topics and difficulties.

- ❖ I added the ability to review quizzes that have just been completed.
  - What it does: This feature allows users to check their answers after they have completed a quiz.
  - Justification: There is not much use in doing a quiz without knowing what questions you got wrong. With this feature, users can see which questions they have answered incorrectly and find out if they have misunderstandings about the topic.
  - Highlights: While the implementation of this feature is similar to the Quiz feature, the input parsing is different from Quiz feature's implementation.

## Code contributed

Please click this link to see a sample of my code: [\[Functional code\]](#)

## Other contributions

- ❖ Enhancements to existing features:
  - Designed and drew mascot sprites for cosmetic improvement of the application.
  - Designed and drafted mock-ups of the GUI in the early stages of the project.
  - Wrote some questions for the Quiz feature.

## Contributions to the User Guide

We updated the original User Guide with instructions for the enhancements that we have added. The following is an excerpt from our **JavaCake User Guide**, showing the additions I have made for the new quiz and review features.

### Doing a quiz

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

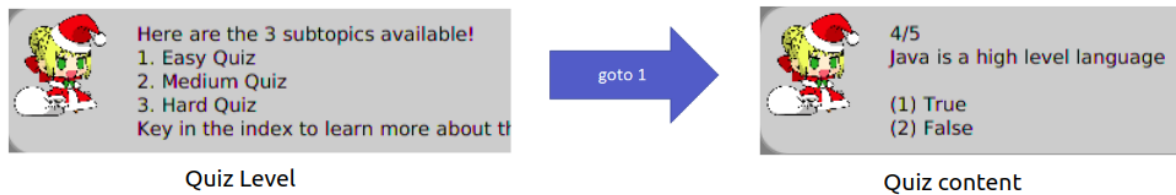
To start a quiz,

1. use `goto` to navigate to any subtopic with the name "Test Yourself!"



*Figure 2: Quiz level displayed*

2. use `goto` to choose your difficulty.



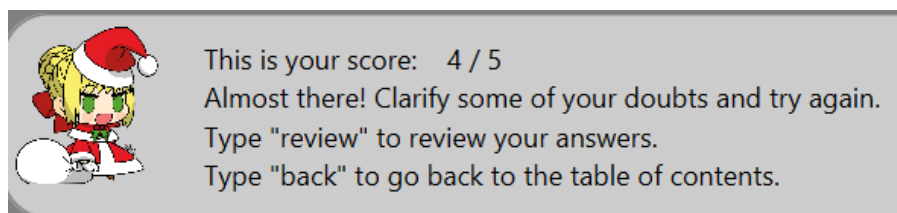
*Figure 3: Quiz launched in the dialogue box*

Alternatively, you can use appended `goto` numberings to jump directly into a quiz as explained in section 3.3. For example, if you want to access the Easy quiz in the Java Basics topic, you may enter `goto 1.4.1`.

The first line indicates the question number and the total number of questions in the quiz.

Each quiz has 5 multiple-choice questions. A question can have between 2 to 5 options, of which only one is the correct answer. Enter the number corresponding to your choice to answer the question. When you have entered your answer, the quiz will move on to the next question, until you have answered all the questions in that quiz.

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.

From the results screen, you may choose to review your answers by typing `review` (refer to Section 3.6) or return to the table of contents by typing `back`.



**All other commands (EXCEPT `exit`) in the program are disabled during the quiz. *JavaCake* will only accept integer inputs during the quiz session.**

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

## Reviewing your answers in a quiz: `review`

After you have completed a quiz, you may check your answers in that quiz by entering `review` on the results page of a quiz.

The review will show a question that you have answered, your answer to that question and the correct answer of that question. To navigate the questions, simply enter the question number. For example, enter “4” to go to question 4.

To quit the review, enter `back` at any point to return to the table of contents.



**`review` can only be entered at the results page of a quiz.**

*JavaCake* does not keep track of previously attempted quiz sessions after you exit the quiz.



**During a review, all commands except `back` are disabled.**

For example, you cannot edit notes while in the middle of a review.



**If you enter an invalid question number, the review will show an error message.**

The review session will also go back to displaying the first question and your answer to that question.

## Contributions to the Developer Guide

The following sections shows my additions the JavaCake Developer Guide for the Model module of the program architecture, as well as the implementation details of the quiz and review features.

### Model (under Design)

The `Model` stores a `QuestionList` object which itself consists of one or more `Question` objects.

`Model` also references `Storage` in order to load and generate the `QuestionList` object from `Storage` when a `QuestionList` object is created.

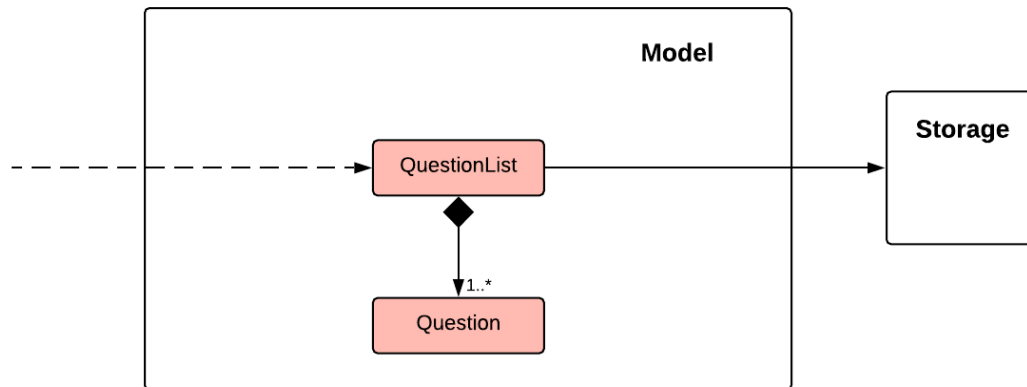


Figure 5: Structure of the Model Component

## Quiz and Review feature

This feature allows users to test themselves on the concepts they have just learned from JavaCoke and review the questions if they wish to do so.

### Proposed Implementation

When `goto [index]` leads to the location of the quiz content, `QuizSession` is called. The question type `qnType` and level of difficulty `qnDifficulty` are both set by reading the `filePath` from `Logic`. A `QuestionList` object, containing an `ArrayList` of `Question` objects, selects `MAX_QUESTIONS` number of questions randomly and loads the corresponding questions from the text files in the directory. `MAX_QUESTIONS` denotes the number of questions in one quiz session. `QuizSession` will use this `QuestionList` object to facilitate the quiz session.

The quiz session starts with a `currScore` of 0. A question is displayed and the program awaits user input. Once user input is received, it is compared to the correct answer of that question and `currScore` increases by 1 if they match. The next question is then displayed to await user input. This continues until all questions in the session are attempted.

When the quiz session is complete, a results screen will be shown to the user, displaying the final `currScore` out of `MAX_QUESTIONS`. Custom messages and different avatar expression will be displayed as well, determined by a `scoreGrade` of `BAD`, `OKAY` or `GOOD`, which in turn is determined by the calculated percentage score in the quiz session. Additionally, 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.

On the results screen, only `review` and `back` are available to be executed. `review` will initialize `ReviewSession`, where the questions from the previously completed quiz session are shown again along with the user's answer and the correct answer. The user

input in a review session should either be integers that correspond to the question number in order to jump to that question, or to execute a `back` command.

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

This feature implements the following methods:

- `QuestionList#loadQuestions()` — loads selected questions in text files and stores them in an `ArrayList`.
- `QuizSession#getQuestion()` — returns the question string.
- `QuizSession#parseInput()` — parses user input. If input is integer sets user's answer to the question, else if the quiz is complete and input is a relevant command returns a response to execute the command.
- `QuizSession#getQuizResult()` — returns the text of the result screen, including *currScore*.
- `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`.
- `ReviewSession#getQuestion()` — returns the question string, along with the user's answer and the correct answer.
- `ReviewSession#parseInput()` — parses user input. If input is integer converts the input to valid index for calling the next question and returns it. If input is `back`, returns response to execute the command.

## Design Considerations

There were several considerations that were made when designing how the quiz feature interacts with storage. The table 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	<code>QuestionList</code> class contains and maintains all hard-coded <code>Question</code> objects and the number of quizzes that each topic contains.  <b>Pros:</b> Easy to implement and test as it is not susceptible to IO or File	Quiz questions are stored with their answers in the text files. <code>QuestionList#loadQuestions()</code> loads the relevant questions from storage.  <b>Pros:</b> Very scalable as additional questions and answers can be easily

	<p>exceptions that may arise from reading from an external file.</p> <p><b>Cons:</b> As all questions and answers have to be hard coded within the class, it is not scalable as number of quiz questions increases.</p>	<p>added without having to manually change the code.</p> <p><b>Cons:</b> Each text file that contains the quiz must follow a certain naming format.</p> <p><b>(Current choice)</b> This method makes it easier to expand quiz content just by adding files without changing the code.</p>
How the quiz questions are randomly selected for a session	<p>Load all questions in the directory, then generate a list of random numbers in the range of the number of questions in the directory. The selected questions are the questions numbers corresponding to the random number list.</p> <p><b>Pros:</b> Easier implementation of <code>QuestionList#loadQuestions()</code> to be less prone to file reading errors caused by incorrect file names.</p> <p><b>Cons:</b> Causes unnecessary read operations for questions that are not chosen and therefore discarded, slowing down performance.</p>	<p>Generate a list of random numbers in the range of the number of questions in the directory first, then load the questions corresponding to the random numbers from the directory.</p> <p><b>Pros:</b> More efficient as it only reads required files, reducing unnecessary read operations.</p> <p><b>Cons:</b> Requires strict naming format of quiz files, prone to errors if quiz files are not numbered correctly.</p> <p><b>(Current choice)</b> This method is more efficient since the program only reads the files that it needs, especially when the size of the question bank increases.</p>