

Plugin writer’s manual

Quiz and Question-type Plugin



October 22, 2018

Priyank joshi

17812234

Table of Contents

[1.0 Introduction 2](#_Toc527706606)

[2.0 Quiz Plugin 2](#_Toc527706607)

[2.1 EngineeringQuiz.java 2](#_Toc527706608)

[3.0 Question-type Plugin 5](#_Toc527706609)

[3.1 ShortAnswer.java 6](#_Toc527706610)

[3.1.1 public ShortAnswer(BlockingQueue<QuestionToUI> bq, ExecutorService es) 6](#_Toc527706611)

[3.1.2 public QuestionType makeShortAnswerQuestion(String question, String answer) 6](#_Toc527706612)

[3.1.3 public boolean isAnswerCorrect() 7](#_Toc527706613)

[3.2 MultiChoice.java 8](#_Toc527706614)

[3.2.1 public MultiChoice(BlockingQueue<QuestionToUI> bq, ExecutorService es) 8](#_Toc527706615)

[3.2.2 public QuestionType makeMultiChoiceQuestion(String question, String[] possibleAnswers, String answer) 9](#_Toc527706616)

[3.2.3 public boolean isAnswerCorrect() 10](#_Toc527706617)

# Introduction

This is a document that explains how to write a Quiz Plugin and Question-type plugin for this application without having to touch, look, or edit the main application code/implementation.

# Quiz Plugin

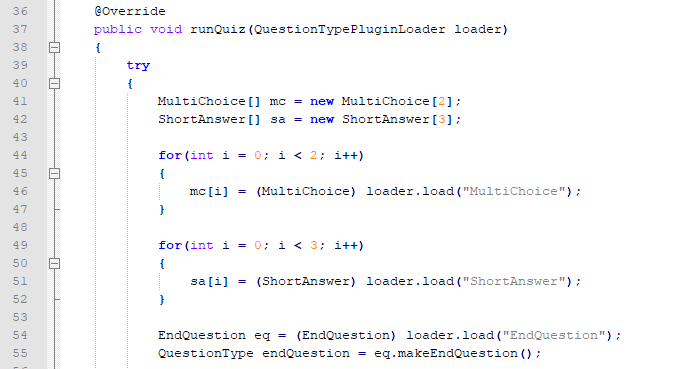
The Quiz plugin contains the structure of a quiz with all the questions and answers. Additionally, all the questions are described in the quiz plugin will be in sequence and displayed to the users correctly. Each quiz plugin represents a complete quiz on a given subject area. For this project, I have a written one quiz plugin – “Engineering Quiz”. You may for example have another type of quiz – “Maths Quiz”, “English Quiz”, “General Knowledge Quiz”, and many more.

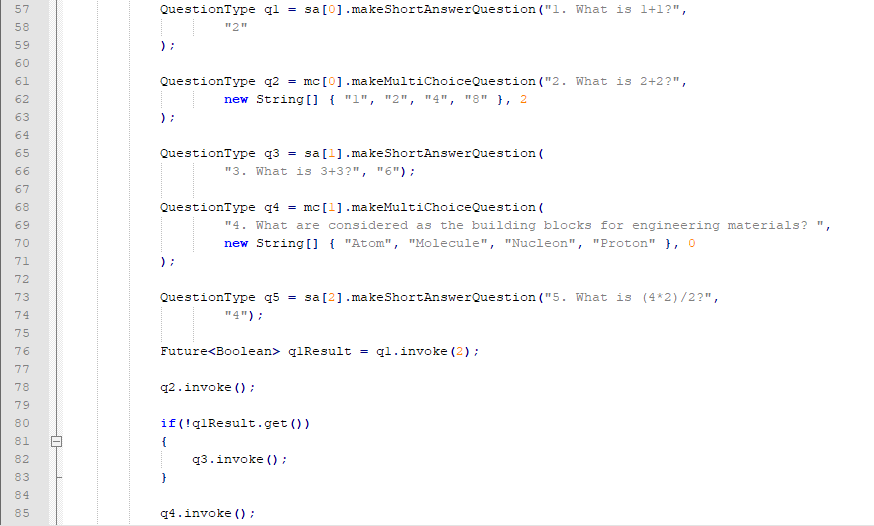
## EngineeringQuiz.java

As mentioned above, this is the only quiz plugin written for this project. This class **MUST** implement ‘QuizPlugin’ (you will not need to know what it contains and how it works).

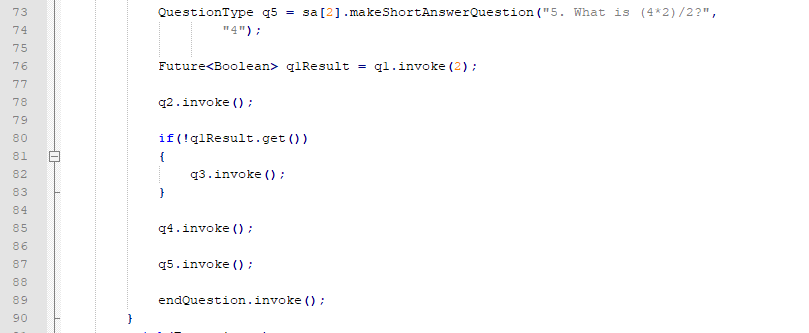
This quiz plugin (EngineeringQuiz.java) will not contain any class fields. Hence, you do not need to specify when creating one.

The quiz plugin contains the methods – which **MUST** have **@Override** before method declarations:

1. public String getName() and returns “Engineering Quiz” – This is essential for the quiz plugin itself as it needs its own name so that when the application loads the quiz, it will display the quiz name.
2. public void runQuiz (QuestionTypePluginLoader loader) – As you are aware of the assignment specification, it is the same method as described in page 2 of the specification.
3. Refer to the picture below for more details.
4. You can load any type of questions that the quiz will have. In the example provided below, I have loaded three types of questions (multi choice, short answer and end question).
5. You **MUST** load the questions based on the number you may want for a particular quiz will have. Refer to the image on page 3.
6. Say for example your quiz has 2 multiple choice questions and 3 short answer questions, then you declare the Question Type Plugin’s -> MultiChoice and ShortAnswer as arrays of type their own respective objects with size (depending on the number you want for each type – in my case I have created size of 3 for short answer because I want to have 3 short answer questions in my quiz). Refer to the image below.
7. The Quiz Plugin **MUST** also load ‘EndQuestion’ (last question of quiz) only once – This is needed so that the application can know when the quiz is ending. Refer to the picture above. We **ONLY** want one end question hence a quiz should never contain two or more end questions.
8. Furthermore, we create a particular question by specifying what kind of question type plugin will handle it.
9. For example: QuestionType q1 = sa[0].makeShortAnswerQuestion(“1. What is 1+1?”, “2”); and so many more questions for a quiz (refer to the image below for more illustration). Note: If we want more short answer questions then it will be sa[1].makeShortAnswerQuestion(“3. What is 3+3?”, “6”);



1. The question type “short answer” makes the question using the method makeShortAnswerQuestion(). This method needs 2 parameters – First is the question itself is ‘What is 1+1?’ and second is answer which is 2.
2. The question type “Multiple choice” makes the question using the method makeMultiChoiceQuestion(). This method needs 3 parameters – First is the question itself is ‘What is 2+2?’, second parameter contains a string array (new String[] {“option1”,”option2”, “etc”}) which consists of options for user to select and third parameter is index specifying the answer which is 2.
3. Then you can call invoke method by invoke() or invoke(30) on every question created. The number 30 for example specifies that the question will have a time limit of 30 seconds. If no time is specified, then the user has unlimited time to answer that question. (The picture above has invoke(2) which means it has a time limit of 2 seconds).
4. If you want to give in a time in the invoke() method then time **MUST** be between 5 and 300 (seconds), if the time specified is less than 5 or greater than 300 seconds then it will throw an exception.
5. Also, if you want conditional questions to occur then you should query for the result of the question via Future<Boolean> object as shown below. For example, if question 1 is right then don’t invoke question 3 and move to question 4 (refer to the image below for further illustration). So, the flow of events is like this attempt question 1, 2, if 1 is wrong then attempt 3, 4, and lastly 5.



1. When you want the quiz to finish then call endQuestion.invoke(); after the last question (as shown above).

# 3.0 Question-type Plugin

This is the type of question that the quiz application will handle. Question-type plugin determines how to display it to the User Interface. I have included two question type plugin’s for this project and will show examples of how I created both them which will help in writing other types of question plugins for example true false question.

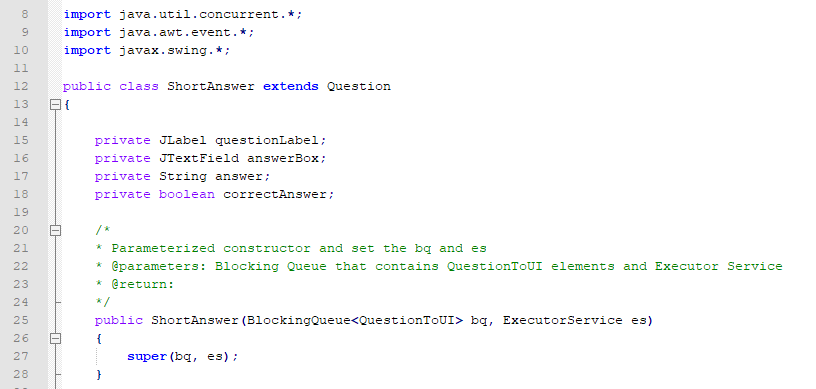
The two-question type plugin’s created which are included in this project are:

1. Short Answer Question
2. Multiple choice Question with 2 or more possible answers

All the question type plugin’s **MUST** extends Question class (refer to the picture below) and has three methods only.

## 3.1 ShortAnswer.java

This type of question requires a question in text (JLabel), simple text input field (JTextField), String answer and boolean correctAnswer for checking if the answer is correct or not.

Hence, we need to declare these as class fields (refer to the image below).

### 3.1.1 public ShortAnswer(BlockingQueue<QuestionToUI> bq, ExecutorService es)

Question type plugin requires a parameterized constructor declared as above and call in super(bq, es) that takes in these parameters (refer to the image above).

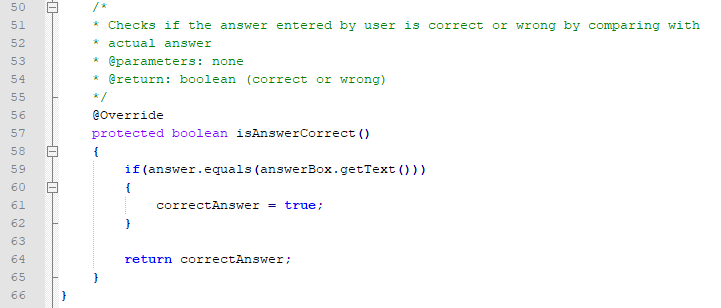
### 3.1.2 public QuestionType makeShortAnswerQuestion(String question, String answer)

This is where you can define how to display the question and it’s contents: answer types (whether text box, multiple choice radio box, etc). But for this type of plugin we will require a simple text input field (JTextField) for inputting answer but question itself is a String.

Each question type plugin will have its own panel. So therefore need to call initialize() method. You will **NEED** to set correctAnswer to false, call setQuestion() method and take in question in string and set answer to the class field. You will then create a label and pass in question string (which we create in Quiz plugin). Additionally, the answer which will be inputted in a text field needs to be instantiated with a column of 10 (minimum space to provide an answer – but you can specify anything more than that in the case if the answer is in sentences and NOT numbers). These components (JTextField and JLabel) will then **NEED** to be added to the panel by calling addToPanel(label) and addToPanel(area) methods. Finally, you can define where you want to set the position of the question and answer box in the panel and return the question itself.

### 3.1.3 public boolean isAnswerCorrect()

This method compares the answer inputted by user with the actual answer provided in the Quiz Plugin (EngineeringQuiz.java – the last parameter) and returns true or false.



## 3.2 MultiChoice.java

The same procedure applies as above but the only difference is displaying the question and its answers – 2 or more possible answers using JRadioButton since this is a different question type (multiple choice question which is completely different from short answer question) and also the last method isAnswerCorrect() since this is a multichoice question (so it needs to check which button is selected, and check the answer based on the selection).

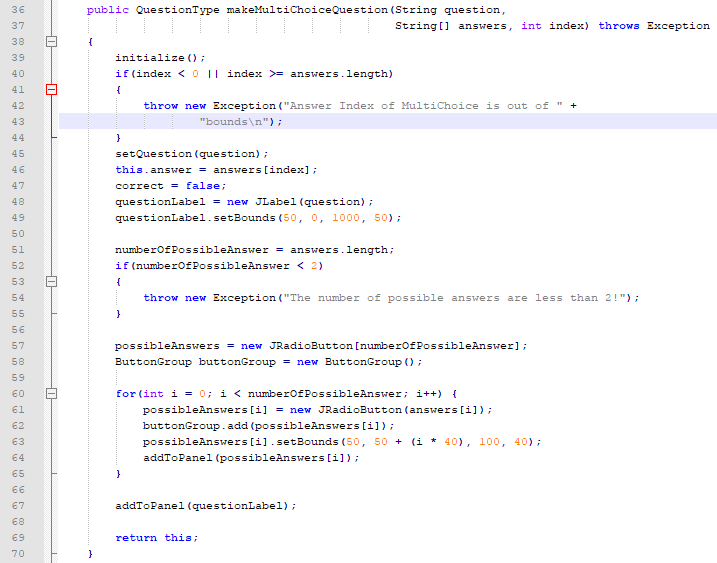
This type of question requires a question in text (JLabel) and a group or array of radio buttons - JRadioButton(according to the assignment specification there should be 2 or more). Additionally, int NumberOfPossibleAnswer, String answer and Boolean correct as class fields.

### 3.2.1 public MultiChoice(BlockingQueue<QuestionToUI> bq, ExecutorService es)

Question type plugin requires a parameterized constructor declared as above and call in super(bq, es) that takes in these parameters.

### 3.2.2 public QuestionType makeMultiChoiceQuestion(String question, String[] possibleAnswers, String answer)

This is where you can define how to display the question and its radio buttons. First initialize the question panel. Then check the condition if index is out of bounds. You will **NEED** to set correct to false, call setQuestion() method and take in question in string and set answer to the to array answers by specifying the index. You will then create a label and pass in question string (which we create in Quiz plugin). Additionally, assign the numberOfPossibleAnswer to the length of the answer array. (minimum number of possible answer(s) are two for this project – that is why we have a condition to check and will throw an exception if the number of possible answers are less than 2). Then create radio buttons based on the number of possible answers. These radio buttons are then added to a group using ButtonGroup as shown in line 62. Then we run a for loop until the number of possible answers to create multiple radio buttons with their possible answers in text, add them to the group, setting their positions in the panel and lastly add them to the panel. Finally, you the panel itself. (refer to the image below for detailed illustration).



### 3.2.3 public boolean isAnswerCorrect()

This method contains a for each loop which will go through each radio button in the array of possible answers radio buttons and check if the selected radio button’s answer in string text is equal to the actual answer (refer to the image on the next page).

