# Plugin Writing FOR DUMMIES®

**Learn to:**

- Write Quiz Plugins
- Write Question Types

**Janek Joyce**

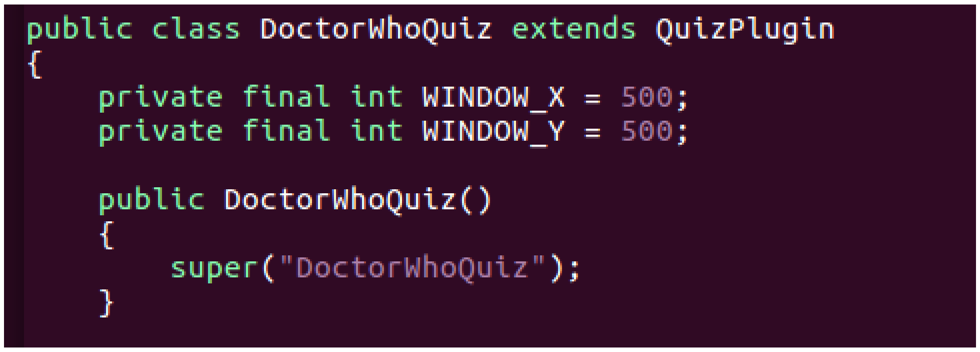
*A Reference for the Rest of Us!*

## Contents

## Quiz Plugins

So, you want to write a quiz plugin for this software? Look no further!

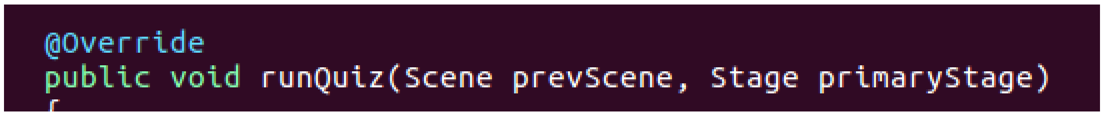
The first step is to create a new blank class with a name of your choice, and extend the abstract class QuizPlugin, as can be seen below:



```java
public class DoctorWhoQuiz extends QuizPlugin
{
    private final int WINDOW_X = 500;
    private final int WINDOW_Y = 500;

    public DoctorWhoQuiz()
    {
        super("DoctorWhoQuiz");
    }
```

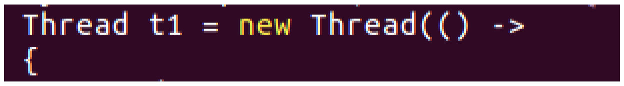
The super class requires you send it the name of this file as the only parameter of the constructor (here it is “DoctorWhoQuiz”), this is so our code can load the class correctly during runtime. The other variables WINDOW_X and WINDOW_Y are not required but make things tidier.

Below is the function where all the fun happens: (Spoiler: You have to implement it)



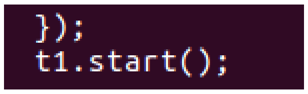
```java
@Override
public void runQuiz(Scene prevScene, Stage primaryStage)
```

The most important part is that you run ALL the following code in its own thread.



```java
Thread t1 = new Thread(() ->
{
```

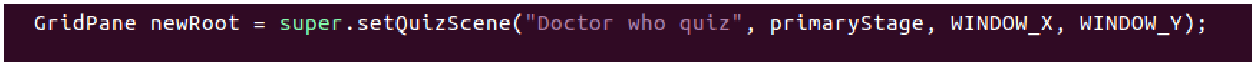
Your code here



```java
});
t1.start();
```

Without this your quiz will not run, it will seem to freeze up.

Next you must specify a new GridPane or similar type of your choosing (if you know what you’re doing), and set up the window with basic information such as the title, window X and window Y resolution. You don’t need to use this function, if you want do to this your own way, feel free to write your own code!



```java
GridPane newRoot = super.setQuizScene("Doctor who quiz", primaryStage, WINDOW_X, WINDOW_Y);
```

Before I move on, you need to keep score! (if you would like to that is), So make an integer someplace handy

```
int myScore = 0;
```

You need to load the plugins that you will be using in your quiz, here I will be loading the Multiple choice and Short answer plugins

```
QuestionType mcp = super.loadPlugin("MultiChoice");
QuestionType sa = super.loadPlugin("ShortAnswer");
```

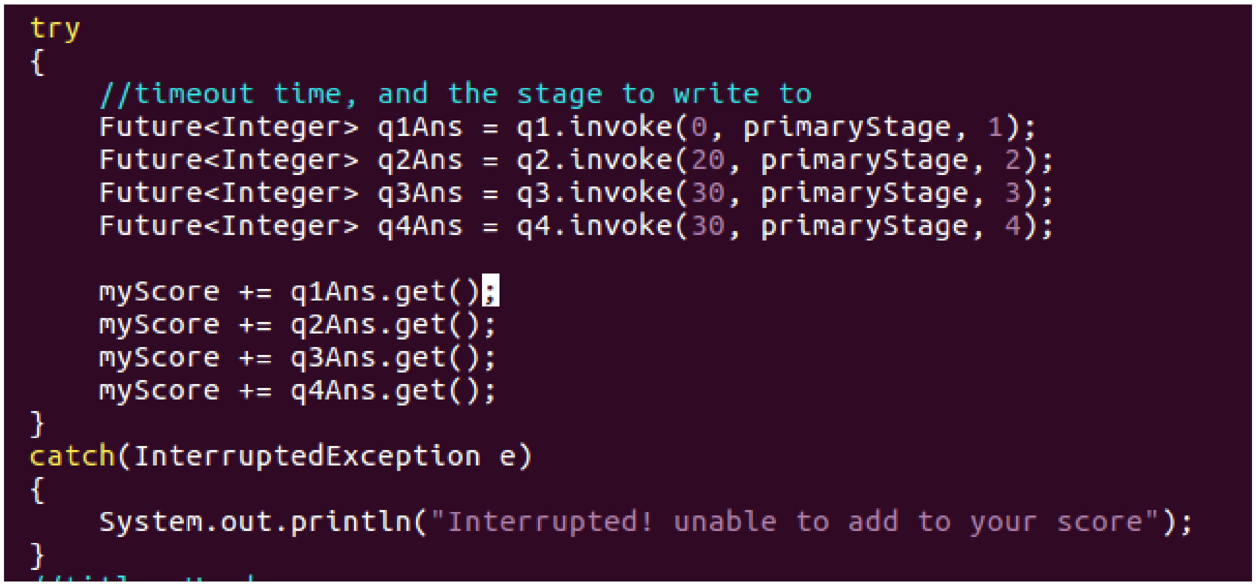
Note: there is also the option of using super.loadPlugins(); (note the ‘s’), this will load ALL plugins in the plugins folder. Then to access these use super.get(“NameOfMyPlugin”);, This method will use less IO relative to loadPlugin() ->if you plan on loading ALL plugins one by one with loadPlugin();.

Now to create your question, (Note: the format is dependent on the question type) for the multiple choice plugin it is; a description, an array of options and the index of the correct answer. The rest is done for you (By the QuestionType). Below you can see 5 different questions,

```
Question q0 = sa.makeQuestion("Are the Daleks robots? Yes/No", "No");
Question q1 = mcp.makeQuestion(
    "Which story did the Doctor first regenerate?",
    new String[] { "The War Games", "Logopolis", "The Tenth Planet", "Planet of the Spiders" },
    2);
Question q2 = mcp.makeQuestion(
    "Which Doctor met the cybermen ONCE?",
    new String[] { "Second", "Third", "Fourth", "Fifth" },
    2);
Question q3 = mcp.makeQuestion(
    "Which is better? Original Doctor who, or NuWho?",
    new String[] { "Original", "New" },
    0);
Question q4 = mcp.makeQuestion("What is 2+2", new String[] { "4", "5", "6" }, 0);
```

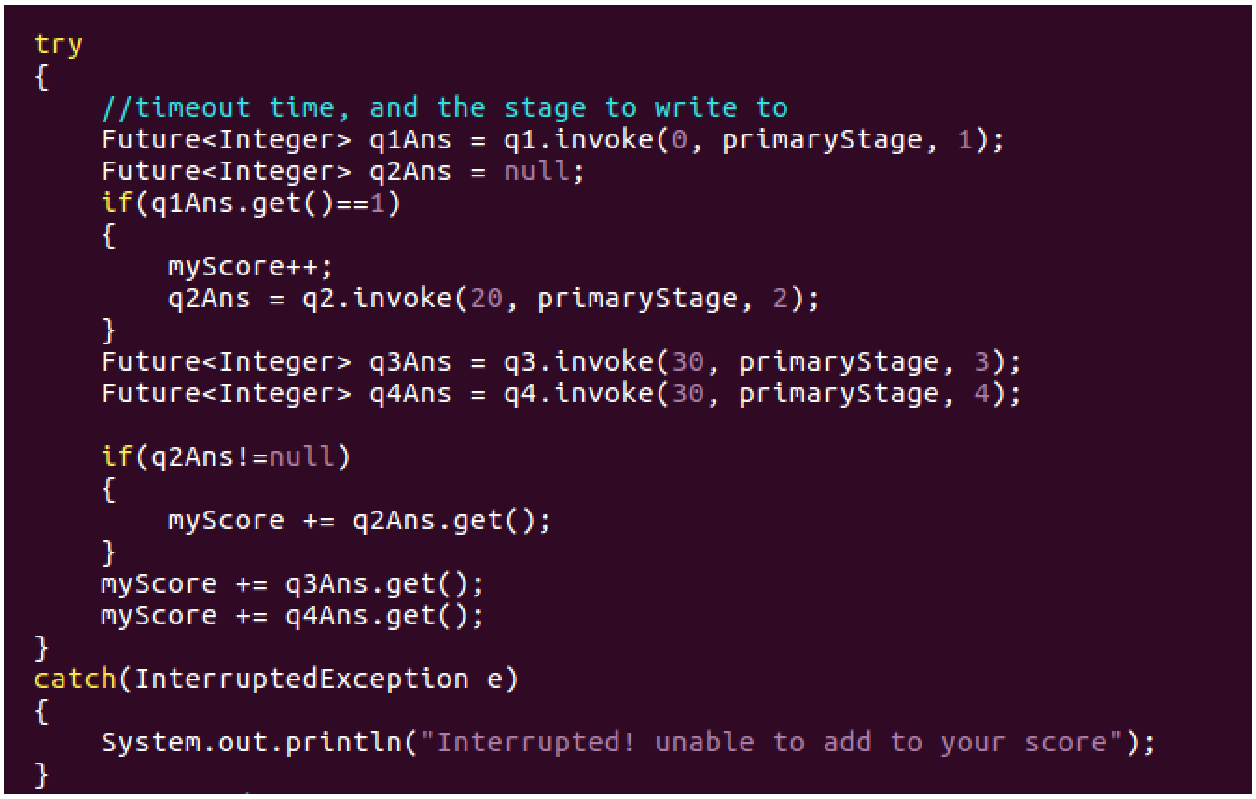
Below is an example of invoking the questions, each one has a timeout value (a value of 0 and below means they have an infinite amount of time to answer the question, any other value is a value in seconds), the primary stage, and a unique question number (Keep it unique!!).

To get the result of the question call question.get(), it will block until the answer is returned.



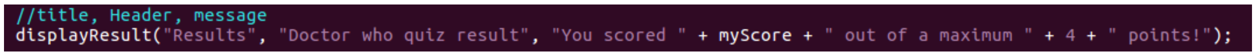
```
try
{
    //timeout time, and the stage to write to
    Future<Integer> q1Ans = q1.invoke(0, primaryStage, 1);
    Future<Integer> q2Ans = q2.invoke(20, primaryStage, 2);
    Future<Integer> q3Ans = q3.invoke(30, primaryStage, 3);
    Future<Integer> q4Ans = q4.invoke(30, primaryStage, 4);

    myScore += q1Ans.get();
    myScore += q2Ans.get();
    myScore += q3Ans.get();
    myScore += q4Ans.get();
}
catch(InterruptedException e)
{
    System.out.println("Interrupted! unable to add to your score");
}
```

A more complicated example can be seen below, the second question will not be shown unless the first question is correct. Its preview will also not be shown.



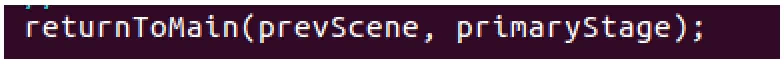
```
try
{
    //timeout time, and the stage to write to
    Future<Integer> q1Ans = q1.invoke(0, primaryStage, 1);
    Future<Integer> q2Ans = null;
    if(q1Ans.get()==1)
    {
        myScore++;
        q2Ans = q2.invoke(20, primaryStage, 2);
    }
    Future<Integer> q3Ans = q3.invoke(30, primaryStage, 3);
    Future<Integer> q4Ans = q4.invoke(30, primaryStage, 4);

    if(q2Ans!=null)
    {
        myScore += q2Ans.get();
    }
    myScore += q3Ans.get();
    myScore += q4Ans.get();
}
catch(InterruptedException e)
{
    System.out.println("Interrupted! unable to add to your score");
}
```

Next is what happens when the quiz is over. Calling this inbuilt function will display the given message in a new window.



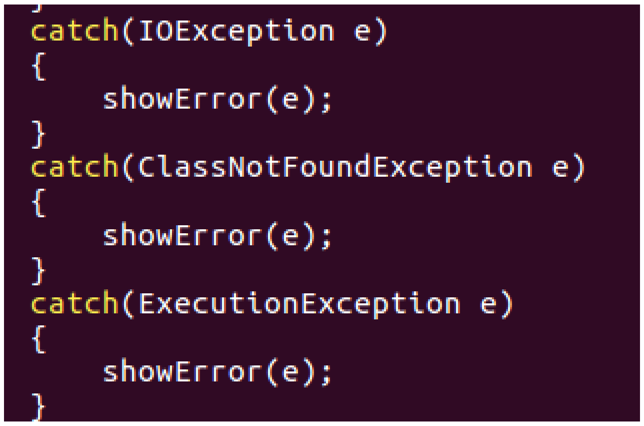
```
//title, Header, message
displayResult("Results", "Doctor who quiz result", "You scored " + myScore + " out of a maximum " + 4 + " points!");
```

And when you are done you will want to return to the main menu! Do so with this:



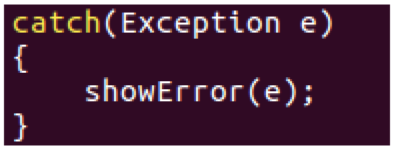
```
returnToMain(prevScene, primaryStage);
```

Finally, you must catch exceptions, you can do them individually



```
catch(IOException e)
{
    showError(e);
}
catch(ClassNotFoundException e)
{
    showError(e);
}
catch(ExecutionException e)
{
    showError(e);
}
```

Or simply



```
catch(Exception e)
{
    showError(e);
}
```

The showError() function is part of the super class, it will show the error in a nice ERROR like way.

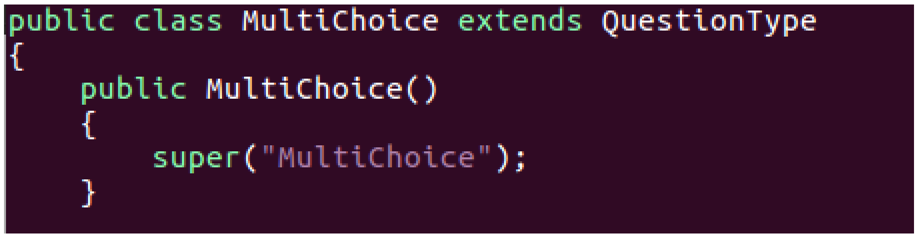
A full example of this code is available in the source files, QuizPlugins/DoctorWhoQuiz/src/main/java/

## IMPORTANT - Plugins of this type need to be in plugins/Quizzes/

## Question Types

Creating a question type is much harder than making a question, you need to know a thing or three about JavaFX here.

The top of your Question type should look something like this:

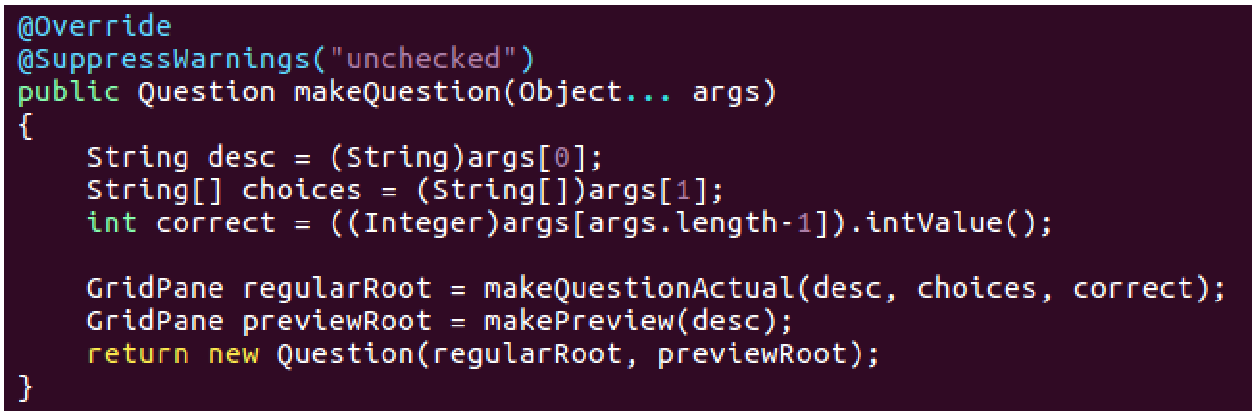


```
public class MultiChoice extends QuestionType
{
    public MultiChoice()
    {
        super("MultiChoice");
    }
```

You must override two functions here, makeQuestion() and makePreview(), these two functions are called to construct the JavaFX parts of your quiz question.

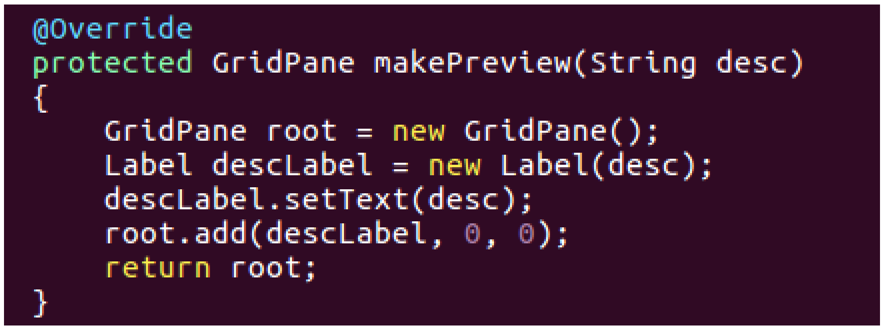
The first, makeQuestion(), takes in an array of Object, this is to allow you (the plugin writer) to have extra control! Here we are sending it the description, choices and the correct index and then constructing the actual question and the preview (they are separate things!).

Finally return a new Question Object.



```
@Override
@SuppressWarnings("unchecked")
public Question makeQuestion(Object... args)
{
    String desc = (String)args[0];
    String[] choices = (String[])args[1];
    int correct = ((Integer)args[args.length-1]).intValue();

    GridPane regularRoot = makeQuestionActual(desc, choices, correct);
    GridPane previewRoot = makePreview(desc);
    return new Question(regularRoot, previewRoot);
}
```

Example makePreview function:



```java
@Override
protected GridPane makePreview(String desc)
{
    GridPane root = new GridPane();
    Label descLabel = new Label(desc);
    descLabel.setText(desc);
    root.add(descLabel, 0, 0);
    return root;
}
```

The actual functionality of the question is up to you (as seen above I call makeQuestionActual() which makes the question), but it MUST have a few things;

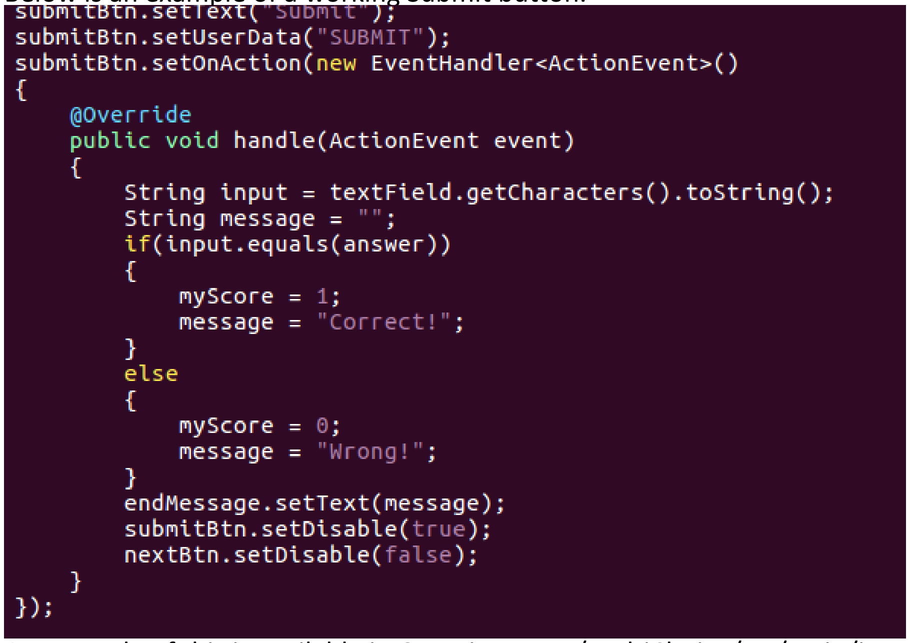
- A Submit button – this checks if your answer is correct, if it is correct you can simply store this value in a variable to use in the:
- Next button - This is when you store the score generated by Submit, you need to place your score in a list already made: GameLogic.score.put(myScore); this will return the result and move onto the next question.

Further information:

- Submit: you MUST have submitBtn.setUserData(“SUBMIT”);
- Next: you MUST have nextBtn.setUserData(“NEXT”);

This is so when a timeout occurs we can squeeze out the unsubmitted result and update their score.

Below is an example of a working Submit button.



```java
submitBtn.setText("Submit");
submitBtn.setUserData("SUBMIT");
submitBtn.setOnAction(new EventHandler<ActionEvent>()
{
    @Override
    public void handle(ActionEvent event)
    {
        String input = textField.getCharacters().toString();
        String message = "";
        if(input.equals(answer))
        {
            myScore = 1;
            message = "Correct!";
        }
        else
        {
            myScore = 0;
            message = "Wrong!";
        }
        endMessage.setText(message);
        submitBtn.setDisable(true);
        nextBtn.setDisable(false);
    }
});
```

An example of this is available in QuestionTypes/MultiChoice/src/main/java/

IMPORTANT - Plugins of this type need to be in plugins/QuestionTypes/