Criterion C: Development

The product is a Java program. Its primary function is in creating a random quiz, the length of which is defined by the user, based on a series of biology definitions. It is also able to print for the user the entire database of questions and answers, from which they can revise.

List of Techniques

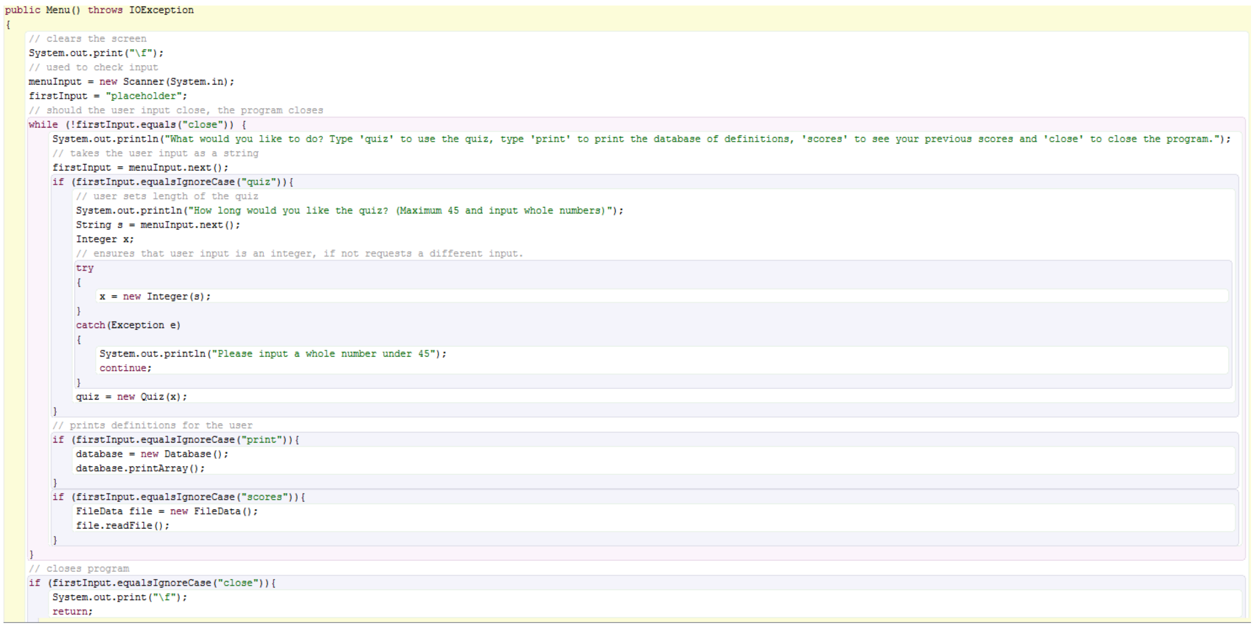
- Random Number Generation
- Evaluating random output
- Parameter passing
- Evaluating user input
- Arrays
- Data validation
- Error handling
- Reading a text file
- Writing to a text file

Program Structure

The highest level of the program is the ‘menu’ class. This class processes the user input that states how they wish to use the program.

The quiz class then creates a series of question objects with information randomly retrieved from the ‘database’ class. It prints it and requests an answer from the user, which it then checks. At the end, it prints the score of that quiz.

This structure was used to make the functionality of the program clear, both for the user and in development. It allowed me to focus first on the creation of the question database class, and its usage, and then to focus on the quiz itself.



```java
public Menu() throws IOException
{
    // clears the screen
    System.out.print("\f");
    // used to check input
    menuInput = new Scanner(System.in);
    firstInput = "placeholder";
    // should the user input close, the program closes
    while (!firstInput.equals("close")) {
        System.out.println("What would you like to do? Type 'quiz' to use the quiz, type 'print' to print the database of definitions, 'scores' to see your previous scores and 'close' to close the program.");
        // takes the user input as a string
        firstInput = menuInput.next();
        if (firstInput.equalsIgnoreCase("quiz")){
            // user sets length of the quiz
            System.out.println("How long would you like the quiz? (Maximum 45 and input whole numbers)");
            String s = menuInput.next();
            Integer x;
            // ensures that user input is an integer, if not requests a different input.
            try
            {
                x = new Integer(s);
            }
            catch(Exception e)
            {
                System.out.println("Please input a whole number under 45");
                continue;
            }
            quiz = new Quiz(x);
        }
        // prints definitions for the user
        if (firstInput.equalsIgnoreCase("print")){
            database = new Database();
            database.printArray();
        }
        if (firstInput.equalsIgnoreCase("scores")){
            FileData file = new FileData();
            file.readFile();
        }
    }
    // closes program
    if (firstInput.equalsIgnoreCase("close")){
        System.out.print("\f");
        return;
```

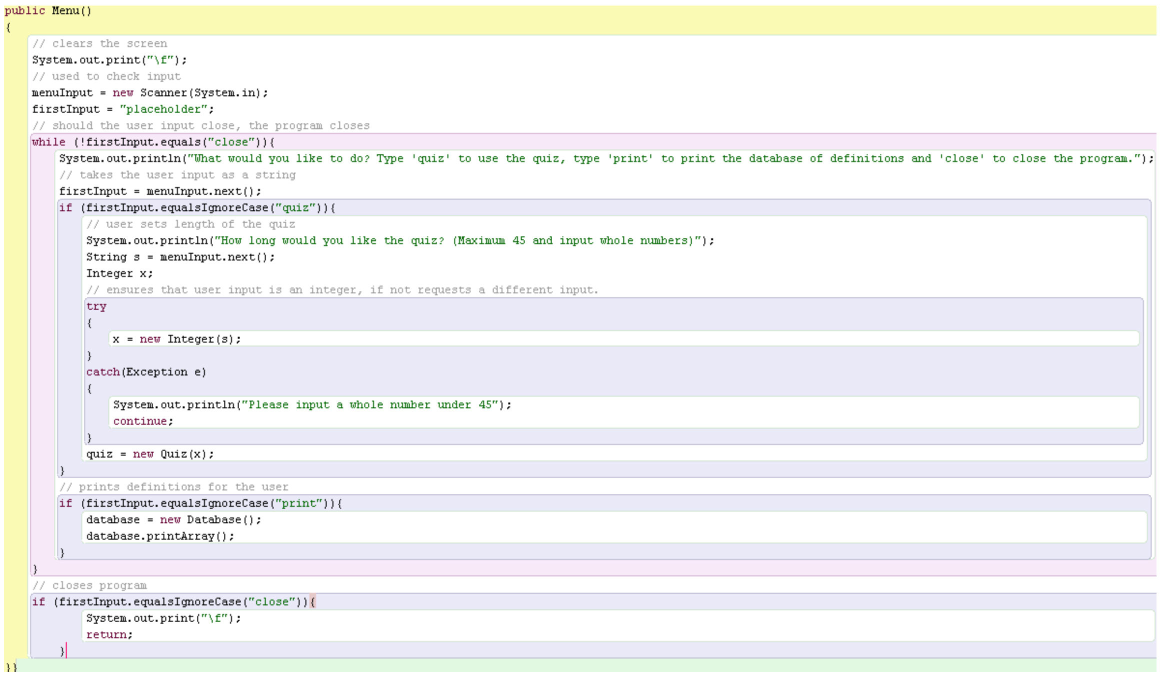
Algorithms

Algorithm	Purpose	Comment
Checking menu input and creating a quiz	Ensuring that menu input is valid, and that the program can act on it	
Creating a random and non-repetitive quiz	Allows the creation of a random question, drawn from a database.	Must ensure no repeats. Draw question and corresponding answer
Evaluating an answer	Checks the users response to that question, giving feedback	
Running the quiz	Ensures that quiz ends in the right place, returns user score	
Saving Scores	Persistently saves scores	Added last following client consultation

Checking menu input and creating a quiz

The menu uses text based navigation, instructing the user what to input, before waiting for the exact commands to continue. Should the user input ‘print’, the program will create a new database class and use the ‘printArray’ method within that class. The quiz itself runs when the user inputs ‘quiz’, after stipulating the length of the quiz, by creating a new quiz object. Should the user input ‘scores’, the score is printed using the FileReader method.

Should the user input ‘close’ at any time, the loop ends and the program stops running.



```java
public Menu()
{
    // clears the screen
    System.out.print("\f");
    // used to check input
    menuInput = new Scanner(System.in);
    firstInput = "placeholder";
    // should the user input close, the program closes
    while (!firstInput.equals("close")){
        System.out.println("What would you like to do? Type 'quiz' to use the quiz, type 'print' to print the database of definitions and 'close' to close the program.");
        // takes the user input as a string
        firstInput = menuInput.next();
        if (firstInput.equalsIgnoreCase("quiz")){
            // user sets length of the quiz
            System.out.println("How long would you like the quiz? (Maximum 45 and input whole numbers)");
            String s = menuInput.next();
            Integer x;
            // ensures that user input is an integer, if not requests a different input.
            try
            {
                x = new Integer(s);
            }
            catch(Exception e)
            {
                System.out.println("Please input a whole number under 45");
                continue;
            }
            quiz = new Quiz(x);
        }
        // prints definitions for the user
        if (firstInput.equalsIgnoreCase("print")){
            database = new Database();
            database.printArray();
        }
    }
    // closes program
    if (firstInput.equalsIgnoreCase("close")){
            System.out.print("\f");
            return;
        }
}}
```

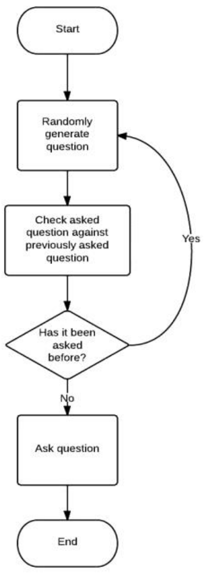
Initially, when requesting an integer as the input, problems arose when the program stopped running if the user inputted a number with a decimal or text, it being the wrong data type. Thus the algorithm had to be altered to first accept the input as a string and then attempt to alter its type to an integer. If it cannot accept it, it requests a valid input from the user.

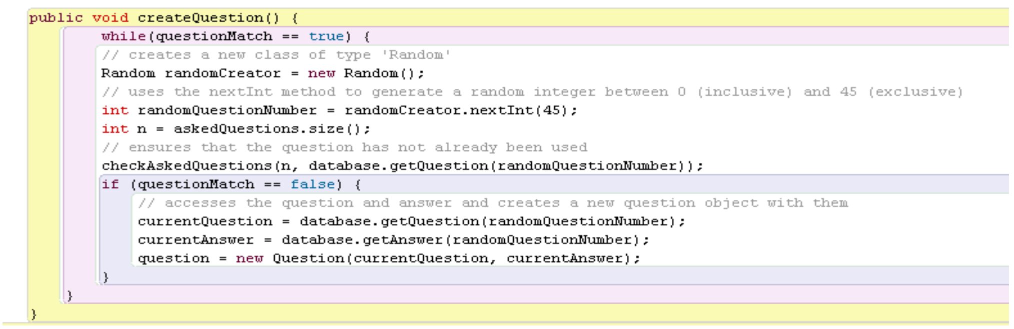
This can be represented through pseudo-code:

```
while INPUT !== 'close'
        print OPTIONS
        request INPUT
        if INPUT == 'quiz'
                request LENGTH
                        if LENGTH can be made an integer
                                create quiz of size LENGTH
                        else request valid input
        if INPUT == 'print'
                print DATABASE.ARRAYS
if INPUT == 'close'
        end
```

<u>Creating a ‘random’ and non repetitive quiz</u>

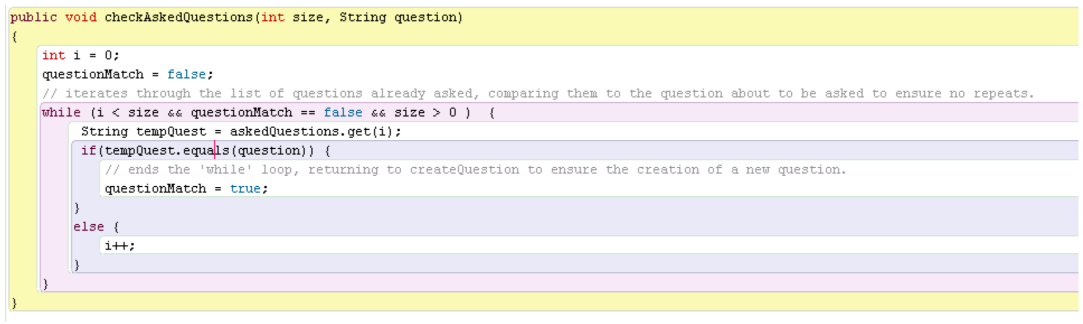
For each question, I decided to use a pseudo-random number generator object within the quiz class to call upon a different field within the arrays. This can be seen in the create question method below.


Start
Randomly generate question
Check asked question against previously asked question
Has it been asked before?
Yes
No
Ask question
End



```
public void createQuestion() {
    while(questionMatch == true) {
    // creates a new class of type 'Random'
    Random randomCreator = new Random();
    // uses the nextInt method to generate a random integer between 0 (inclusive) and 45 (exclusive)
    int randomQuestionNumber = randomCreator.nextInt(45);
    int n = askedQuestions.size();
    // ensures that the question has not already been used
    checkAskedQuestions(n, database.getQuestion(randomQuestionNumber));
    if (questionMatch == false) {
        // accesses the question and answer and creates a new question object with them
        currentQuestion = database.getQuestion(randomQuestionNumber);
        currentAnswer = database.getAnswer(randomQuestionNumber);
        question = new Question(currentQuestion, currentAnswer);
    }
    }
}
```

In this method, the ‘randomQuestionNumber’ variable (between 0 and 44 inclusive) generated by the java ‘Random’ class is used to ‘get’ a question variable from the database. This is in order to evaluate whether the question had already been asked through in this particular quiz. . It does this by checking the question against a list of already asked questions, created as the ‘quiz’ class runs, in the checkAskedQuestions method (seen below). If it finds that the question has been asked, it returns to the top of the algorithm and creates another random question.



```
public void checkAskedQuestions(int size, String question)
{
    int i = 0;
    questionMatch = false;
    // iterates through the list of questions already asked, comparing them to the question about to be asked to ensure no repeats.
    while (i < size && questionMatch == false && size > 0 )  {
        String tempQuest = askedQuestions.get(i);
        if(tempQuest.equals(question)) {
            // ends the 'while' loop, returning to createQuestion to ensure the creation of a new question.
            questionMatch = true;
        }
        else {
            i++;
        }
    }
}
```

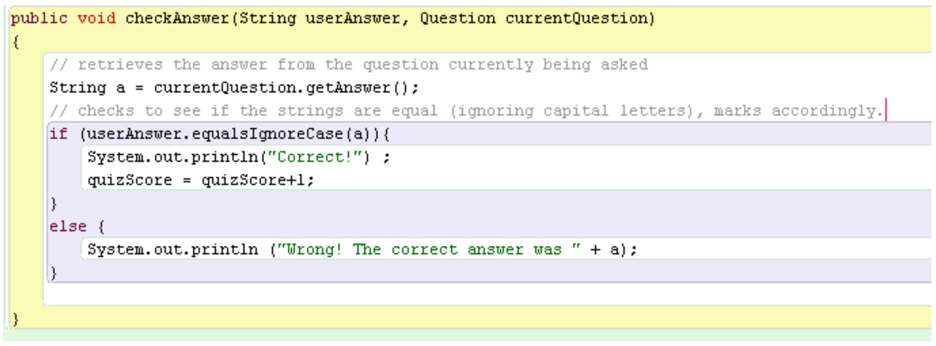
It is able to do this by altering the ‘questionMatch’ Boolean variable. This variable is used to control the actions of the ‘createQuestions’ method. In the constructor of class ‘quiz’, and each time before createQuestions is used, this variable is set to ‘true’ - ensuring that the ‘createQuestions’ method creates a new random question and checks whether it has been asked previously. In checkAskedQuestions, questionMatch is then set to false, and only

becomes true if the random output is evaluated to match the list of already asked questions.
If questionMatch stays false, a new question object is created.

<u>Evaluating an answer</u>

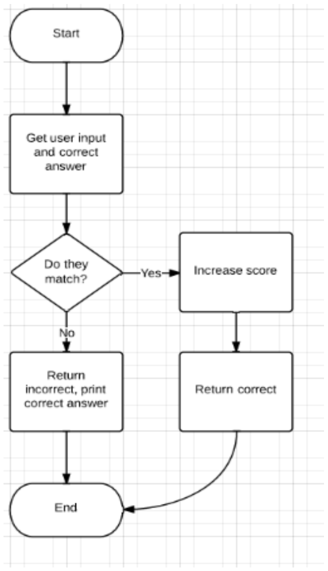
This algorithm is again held within the quiz class, as the ‘checkAnswer’ method, shown below, which takes the users input as its parameters.

Simply, this algorithm takes the user answer and retrieves the correct answer from the current question and checks to see if they match. If they do it increases the user’s score and prints ‘correct’, if not it tells the user that they were wrong and prints the correct answer.



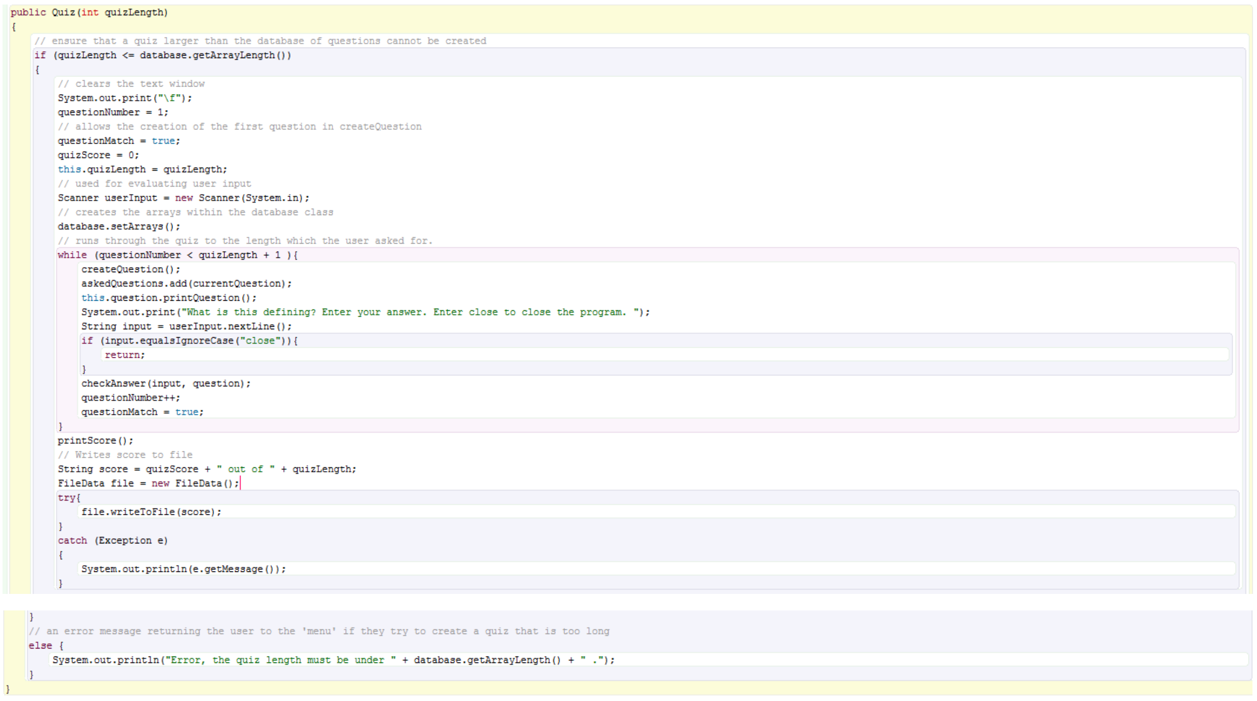
```
public void checkAnswer(String userAnswer, Question currentQuestion)
{
    // retrieves the answer from the question currently being asked
    String a = currentQuestion.getAnswer();
    // checks to see if the strings are equal (ignoring capital letters), marks accordingly.
    if (userAnswer.equalsIgnoreCase(a)){
        System.out.println("Correct!") ;
        quizScore = quizScore+1;
    }
    else {
        System.out.println ("Wrong! The correct answer was " + a);
    }

}
```


Start
Get user input and correct answer
Do they match?
Yes
Increase score
No
Return incorrect, print correct answer
Return correct
End

<u>Running the quiz</u>

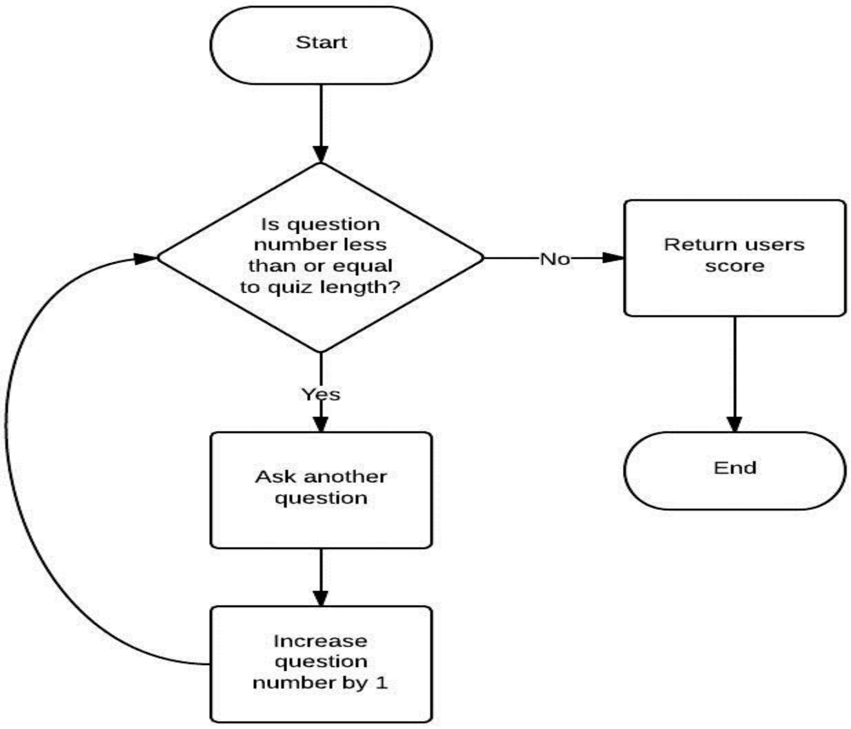
This Ensures that the quiz is the correct length, and ends at the right point. It also includes the writing of the text to a file Seen here in constructor of the 'quiz' class:



```java
public Quiz(int quizLength)
{
    // ensure that a quiz larger than the database of questions cannot be created
    if (quizLength <= database.getArrayLength())
    {
        // clears the text window
        System.out.print("\f");
        questionNumber = 1;
        // allows the creation of the first question in createQuestion
        questionMatch = true;
        quizScore = 0;
        this.quizLength = quizLength;
        // used for evaluating user input
        Scanner userInput = new Scanner(System.in);
        // creates the arrays within the database class
        database.setArrays();
        // runs through the quiz to the length which the user asked for.
        while (questionNumber < quizLength + 1 ){
            createQuestion();
            askedQuestions.add(currentQuestion);
            this.question.printQuestion();
            System.out.print("What is this defining? Enter your answer. Enter close to close the program. ");
            String input = userInput.nextLine();
            if (input.equalsIgnoreCase("close")){
                return;
            }
            checkAnswer(input, question);
            questionNumber++;
            questionMatch = true;
        }
        printScore();
        // Writes score to file
        String score = quizScore + " out of " + quizLength;
        FileData file = new FileData();
        try{
            file.writeToFile(score);
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }

    }
    // an error message returning the user to the 'menu' if they try to create a quiz that is too long
    else {
        System.out.println("Error, the quiz length must be under " + database.getArrayLength() + " .");
    }
}
```

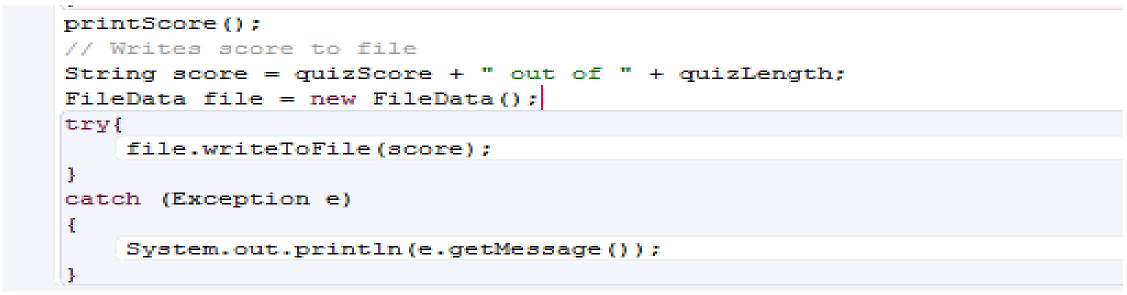
Diagrammatically, this can simply be shown like so:



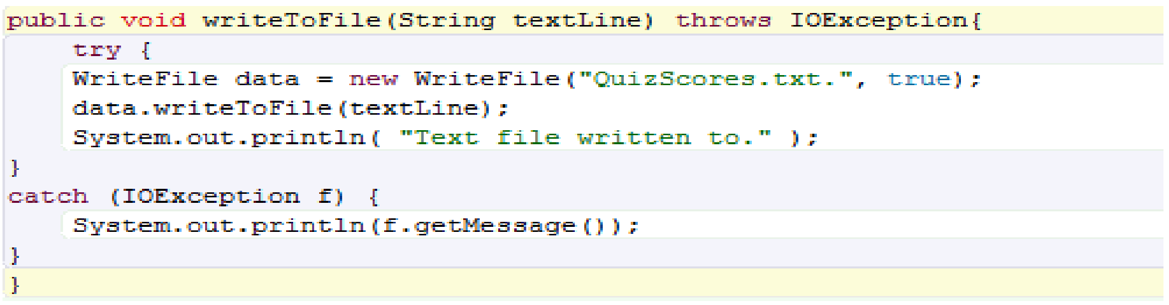
Start
Is question number less than or equal to quiz length?
No
Return users score
End
Yes
Ask another question
Increase question number by 1

Saving Scores[1]

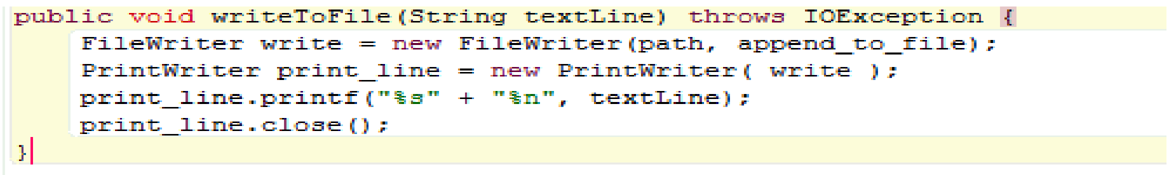
Added last, this takes the score at the end of each quiz and adds it to a text file, which can then be printed from the menu. For this I used the java FileWriter classes, designed for this function, and had the write method completed at the end of the quiz, like so:



```
printScore();
// Writes score to file
String score = quizScore + " out of " + quizLength;
FileData file = new FileData();
try{
    file.writeToFile(score);
}
catch (Exception e)
{
    System.out.println(e.getMessage());
}
```



```
public void writeToFile(String textLine) throws IOException{
    try {
    WriteFile data = new WriteFile("QuizScores.txt.", true);
    data.writeToFile(textLine);
    System.out.println( "Text file written to." );
}
catch (IOException f) {
    System.out.println(f.getMessage());
}
}
```



```
public void writeToFile(String textLine) throws IOException {
    FileWriter write = new FileWriter(path, append_to_file);
    PrintWriter print_line = new PrintWriter( write );
    print_line.printf("%s" + "%n", textLine);
    print_line.close();
}
```

As this was added late in the process, I have drawn up a separate test plan for it:

Test Number	Description	Input Data/Instructions	Expected Results
8	Write to file	Complete a quiz. open the QuizScores.txt file.	The message 'Text file written to' should be printed. The file should contain the score achieved.
9	Print Scores – read file	Type in 'scores' on the menu	The previous scores achieved should be printed.

Data Structures

I used two arrays within the 'database' class to store the questions and answers and an 'arrayList' within the quiz class to hold the questions that had already been asked as the program was running. The two arrays within the database class were designed to correspond numerically (the data held at [0] in the answers array corresponds to that at [0] at questions array) and thus a list would have been inappropriate for this type of storage. Conversely the list used to store the previously asked questions is continually added to while the program is running and changes in length depending upon the user's preferences. It is therefore more efficient if it is a list, with no fixed length, rather than an array.

992 words

Bibliography

Cram. 'Biology Revision Cards'. Last accessed xx.
www.cram.com/flashcards/definitions-for-as-level-biology-chapter-2-1939112

Home and Learn. 'Reading and writing a textfile in Java'. Last accessed xx.
http://www.homeandlearn.co.uk/java/read_a_textfile_in_java.html