Criterion C - 1075 words

Table of Content

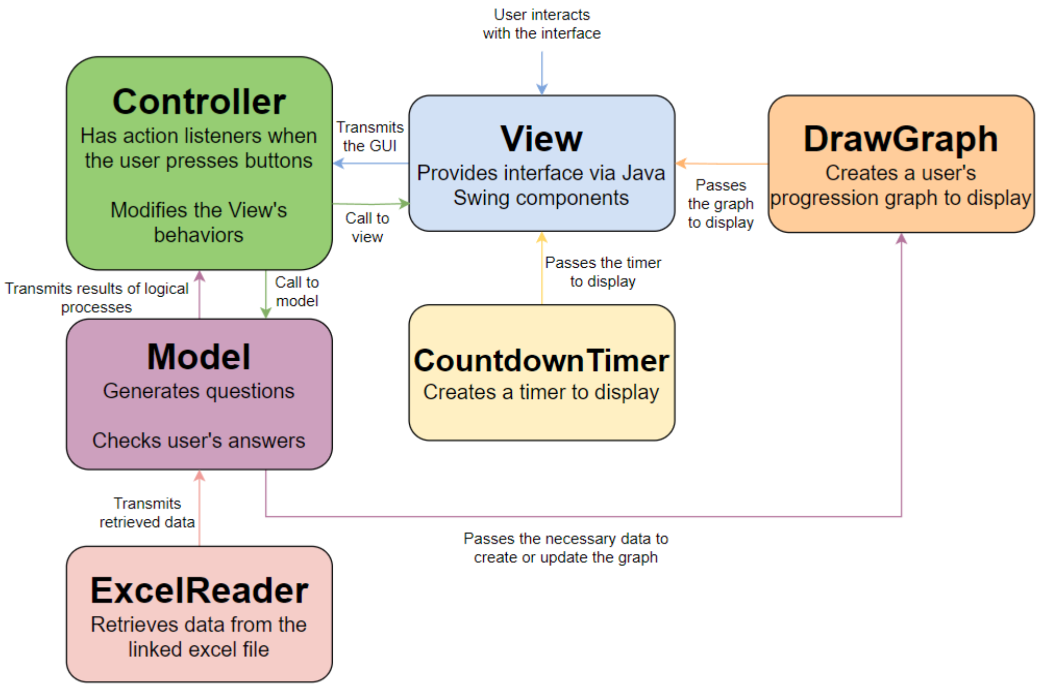
Object Oriented Programming

MVC

I decided to use object-oriented programming for my project, integrating the MVC Model for its structured organization of classes with defined roles and relationships. As such, I embraced its three classes, specifically:

1. **Model**: Handles the logical thinking and algorithms of the program and manipulates data.
2. **View**: Contains all the Swing User Interface components that provide a visual display of data.
3. **Controller**: Has instances of the Model and the View class, controlling their interactions and passing data between them

Document 1: Diagram of relationships and interactions of all classes


User interacts with the interface
Controller
Has action listeners when the user presses buttons
Modifies the View's behaviors
Transmits the GUI
Call to view
View
Provides interface via Java Swing components
Passes the graph to display
DrawGraph
Creates a user's progression graph to display
Transmits results of logical processes
Call to model
Passes the timer to display
Model
Generates questions
Checks user's answers
CountdownTimer
Creates a timer to display
Transmits retrieved data
Passes the necessary data to create or update the graph
ExcelReader
Retrieves data from the linked excel file

Document 2: Documentation of methods in the "Model" class

Method name	Return type	Parameters	Description
Model	/	/	-The constructor method of the Model class that allows one to create instances of the Model class in other classes
retrieveAnswerAText retrieveAnswerBText retrieveAnswerCText retrieveAnswerDText	String	/	-Retrieves the data string of an answer choice from the Excel file
checkAnswer	boolean	/	-Checks if the user's answer is correct -Updates the variable "correctAnswer", collections "wrongAnswerRows" and "areaToFocus" when necessary
getQuestionRowDomain	ArrayList<Integer>	/	-Gets the domain of rows of a certain study topic in the Excel file
questionProvider	String	/	-Retrieves a random question from the Excel file according to the domain of the rows
checkWrongTopicOccurences	ArrayList<String>	/	-Generates the study topic the user is weak at
updateUserScores	void	/	-Updates the scores the user got from all previous quizzes

Document 3: Documentation of methods in the "View" class

Method name	Return type	Parameters	Description
View	/	/	-The constructor method of the View class that allows one to create instances of the View

			class in other classes
addStartButtonListener addNextButtonListener addStartPracticeQuestionsListener addTerminateButtonListener addAnswerAListener addAnswerBListener addAnswerCListener addAnswerDListener addQuizDurationDropBoxListener addKinematicsListener addForcesAndMomentumListener AddWorkPowerEnergyListener addRotationalMechanicsListener addThermalEnergyTransfersListener addGreenhouseEffectListener addIdealGasLawsListener addThermodynamicsListener	void	**ActionListener** listener	-Adds an **actionListener** called “listener” to their respective attribute -Allows these buttons, ComboBox, and radioButtons to be created in the Controller class that define specific actions to implement when the Swing button element is activated
displayHomePage	void	/	-Displays the home page and deactivates all other pages
displayQuestionPage	void	/	-Displays the question page and deactivates all other pages
displayResultPage	void	/	-Displays the result page and deactivates all other pages
setQuestionTextArea		/	-**Mutator**: changes the text being displayed in the question area
setAnswerAText setAnswerBText setAnswerCText setAnswerDText	void	/	-**Mutator**: changes the text being displayed in each of the answer choices

setResultAreaText	void	/	-**Mutator**: changes the text being displayed in the score area
setTopicsToFocusOnText	void	/	-**Mutator**: changes the text being displayed in the topics to focus on area
answerChoiceAPicked answerChoiceBPicked answerChoiceCPicked answerChoiceDPicked	void	/	-Changes the answer choice buttons' behaviors when clicked

Document 4: Documentation of methods in the "Controller" class

Method name	Return type	Parameters	Description
Controller	/	/	-The constructor method of the Controller class that allows one to create instances of the Controller class in other classes
startQuizButtonListener - actionPerformed	void	**ActionEvent** e	-Initializes timer -Changes GUI page -Retrieves necessary data from Excel file to display
nextButtonListener - actionPerformed	void	**ActionEvent** e	-Resets question page -Checks if the quiz is over -Displays result page if yes -Generates a new question if no
terminateButtonListener	void	**ActionEvent** e	-Resets everything in the program
startPracticeQuestionsListener -	void	**ActionEvent** e	-Displays question page

actionPerformed			-Generates undone questions
quizDurationDropBoxListener - actionPerformed	void	**ActionEvent** e	-Records what duration the user selects -Checks if the necessary requirements are met for the user to start the quiz
kinematicsListener - actionPerformed forcesAndMomentumListener - actionPerformed workPowerEnergyListener - actionPerformed rotationalMechanicsListener - actionPerformed thermalEnergyTransfersListener - actionPerformed greenhouseEffectListener - actionPerformed idealGasLawsListener - actionPerformed thermodynamicsListener - actionPerformed	void	**ActionEvent** e	-Records what study topics the user has selected for the quiz -Checks if the necessary requirements are met for the user to start the quiz
answerAButtonListener - actionPerformed answerBButtonListener - actionPerformed answerCButtonListener - actionPerformed answerDButtonListener - actionPerformed	void	**ActionEvent** e	-Checks the user's selected answer with the correct answer -Modifies the GUI components
main	void	**String[]** args	- Is responsible for initializing instances of all classes that are to be used when the program is run: model, view,

			controller, ExcelReader, CountdownTimer, and drawGraph - the first home page is made visible

Document 5: Documentation of methods in the "ExcelReader" class

Method name	Return type	Parameters	Description
readExcel	String	**String** sheetName **int** rNum **int** cNum	Retrieves a specific data from the Excel file according to the parameters

Libraries

Document 6: Documentation of all the libraries used

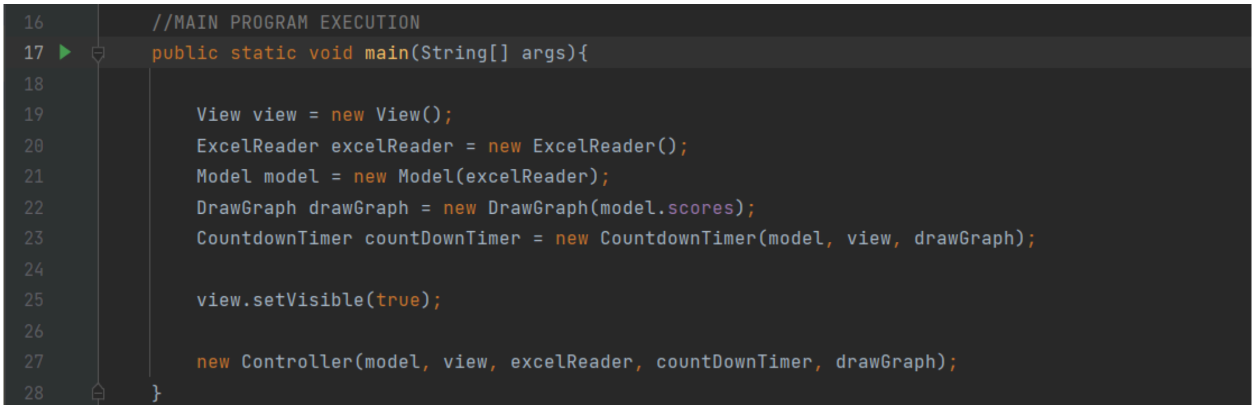
Classes	How it is used
java.awt	-Provides a set of visual elements for creating a graphical user interface, along with the necessary mechanisms for linking user interactions to algorithmic actions. -In this project, the **AWT** class was used to assign behaviors to the **JComponents** upon each user interaction. It is also used to draw and display the progression graph
java.swing	-This is the library used to create and display the user interface with a more sophisticated set of JComponent than **AWT**. -In this project, the **Swing** class was used to display all the GUI components such as **JButton**, **JTextArea**, or **JRadioButton**.
java.io	-This package provides for system input and output through a data stream. -In this project, the **IO** class was used to input the Excel question bank into the program. Then the program would be able to retrieve data from the inputted Excel file.
org.apache.poi	-This library allows Java programs to read and write files in Microsoft Office formats, such as

	Word, PowerPoint and Excel. -In this project, **Apache POI** was used to read data from the Excel question bank file.
java.util	-This library provides static methods that are accessible for use across an application. The static methods are used for performing common routines in our application. -In this project, the **Collections** subclass from the **Util** class was mainly used to store data that would only be defined during program execution.

Algorithmic thinking

How the program starts

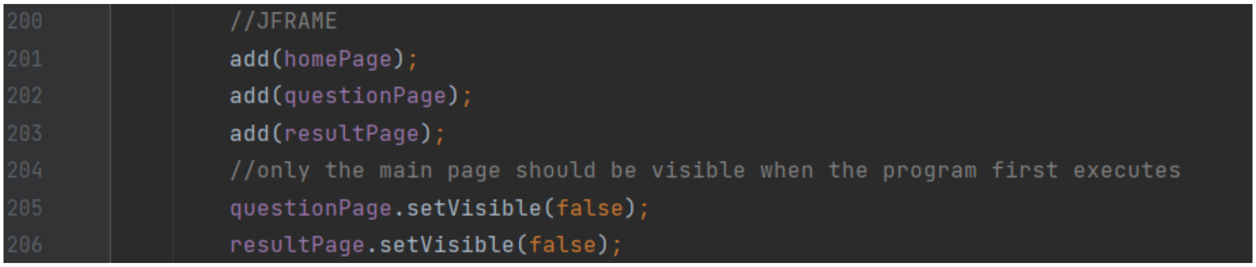
Document 7: Screenshot of the code when the program starts



```
16      //MAIN PROGRAM EXECUTION
17      public static void main(String[] args){
18
19          View view = new View();
20          ExcelReader excelReader = new ExcelReader();
21          Model model = new Model(excelReader);
22          DrawGraph drawGraph = new DrawGraph(model.scores);
23          CountdownTimer countDownTimer = new CountdownTimer(model, view, drawGraph);
24
25          view.setVisible(true);
26
27          new Controller(model, view, excelReader, countDownTimer, drawGraph);
28      }
```

From lines 19 to 23, instances of all classes are initialized. This is because the program can only use the instances of classes and not their own definitions. At line 25, the frames and components from class **view** are made visible onto the GUI screen. And at line 27, an instance of the class **controller** is initialized. This class has parameters of every other class because it is the brain of the program, it controls relationships and communications between classes, just like its role in the MVC.

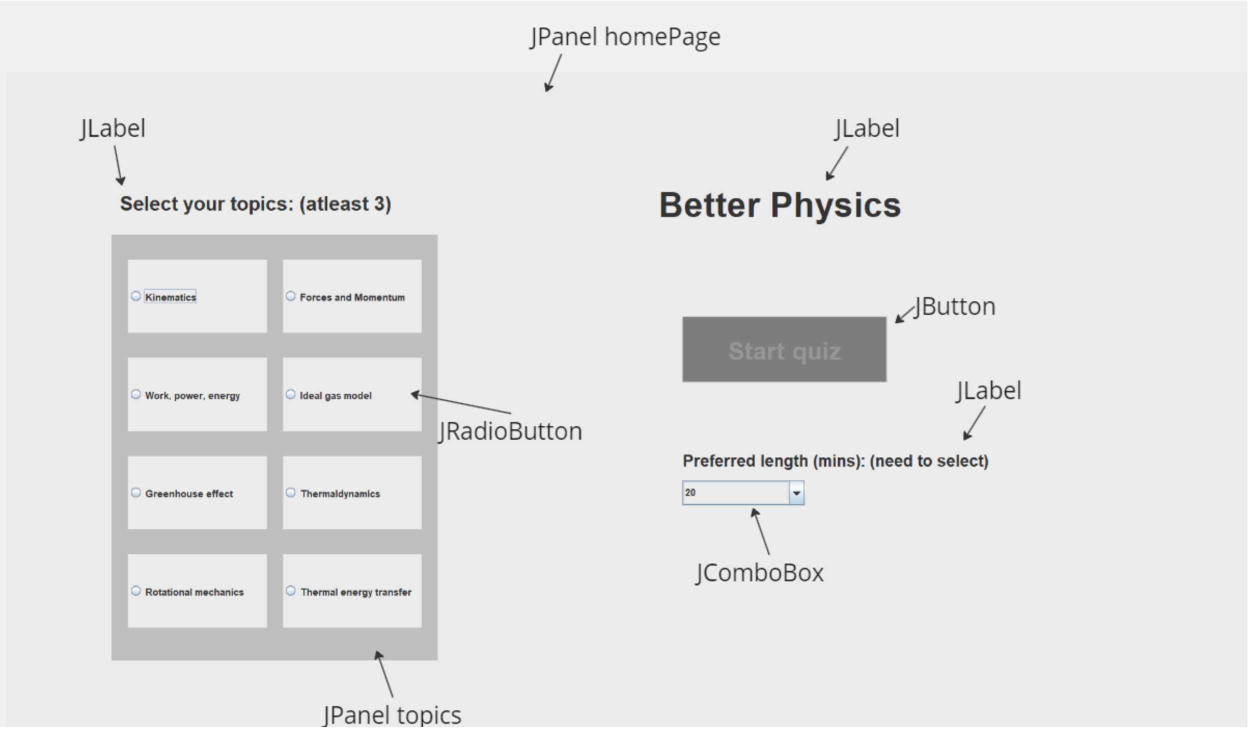
Document 8: Screenshot of the code of when the pages should be displayed or not



```
200         //JFRAME
201         add(homePage);
202         add(questionPage);
203         add(resultPage);
204         //only the main page should be visible when the program first executes
205         questionPage.setVisible(false);
206         resultPage.setVisible(false);
```

Then, in the **view** constructor, only the **homePage JPanel** is being displayed. From lines 201 to 203, the **JPanels** representing the pages are added to the main frame so they can be accessed. The **questionPage** and **resultPage** are set to invisible at lines 205 and 206 because the **homePage** is where the program should begin with.

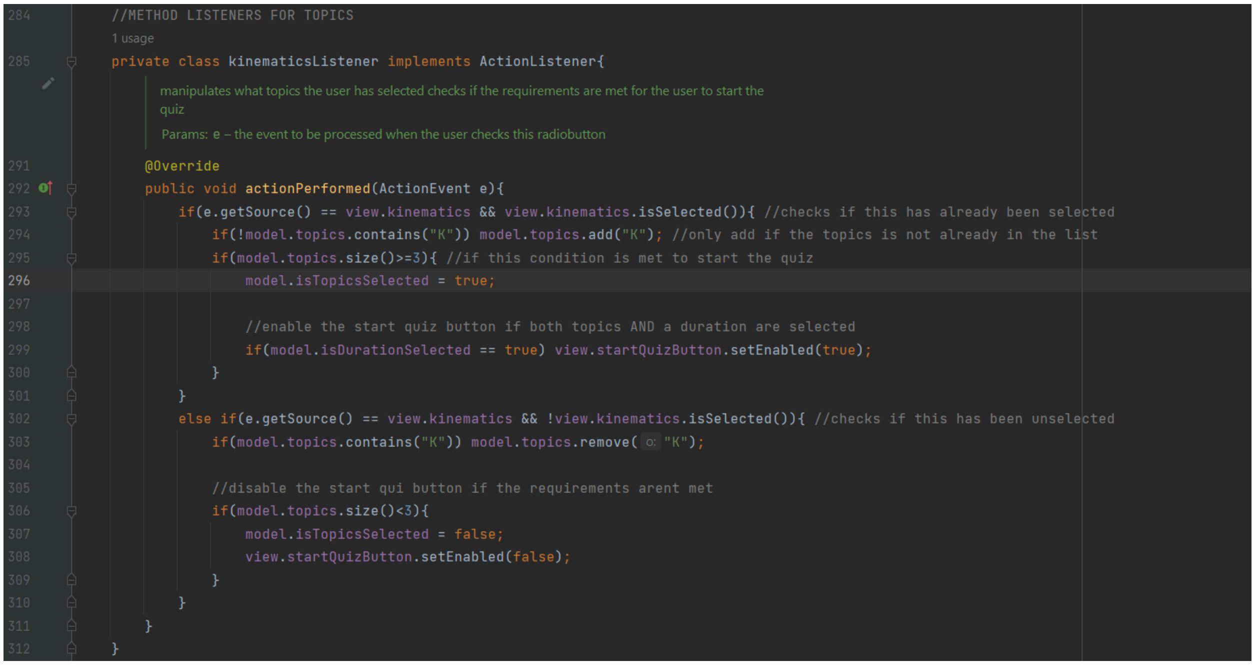
Document 9: Labeling the home page with the GUI components



1. **JLabel**: used to display a short string at a defined position
2. **JButton**: used to interact with the user and to navigate the GUI
3. **JComboBox**: used for the user to select an option from a dropdown menu
4. **JRadioButton**: used for the user to select multiple options
5. **JPanel:** used to organize each page to display

Requirements to start the quiz

Document 10: Screenshot of the code of one of the RadioButtons



```
284     //METHOD LISTENERS FOR TOPICS
        1 usage
285     private class kinematicsListener implements ActionListener{
            manipulates what topics the user has selected checks if the requirements are met for the user to start the
            quiz
            Params: e – the event to be processed when the user checks this radiobutton
291         @Override
292         public void actionPerformed(ActionEvent e){
293             if(e.getSource() == view.kinematics && view.kinematics.isSelected()){ //checks if this has already been selected
294                 if(!model.topics.contains("K")) model.topics.add("K"); //only add if the topics is not already in the list
295                 if(model.topics.size()>=3){ //if this condition is met to start the quiz
296                     model.isTopicsSelected = true;
297
298                     //enable the start quiz button if both topics AND a duration are selected
299                     if(model.isDurationSelected == true) view.startQuizButton.setEnabled(true);
300                 }
301             }
302             else if(e.getSource() == view.kinematics && !view.kinematics.isSelected()){ //checks if this has been unselected
303                 if(model.topics.contains("K")) model.topics.remove( o: "K");
304
305                 //disable the start qui button if the requirements arent met
306                 if(model.topics.size()<3){
307                     model.isTopicsSelected = false;
308                     view.startQuizButton.setEnabled(false);
309                 }
310             }
311         }
312     }
```

6. **ActionListener:** defines what the program does following an user interaction

At line 294, the topic represented by the **JRadioButton** is added to an **ArrayList** storing all the study topics selected by the user. From lines 295 to 300, I check if the user has selected at least 3 topics AND a quiz duration, only then the user can start the quiz. I have decided that at least 3 topics should be selected because of an extreme case where the number of questions from one topic isn't sufficient for a certain quiz duration if it is the only topic selected.

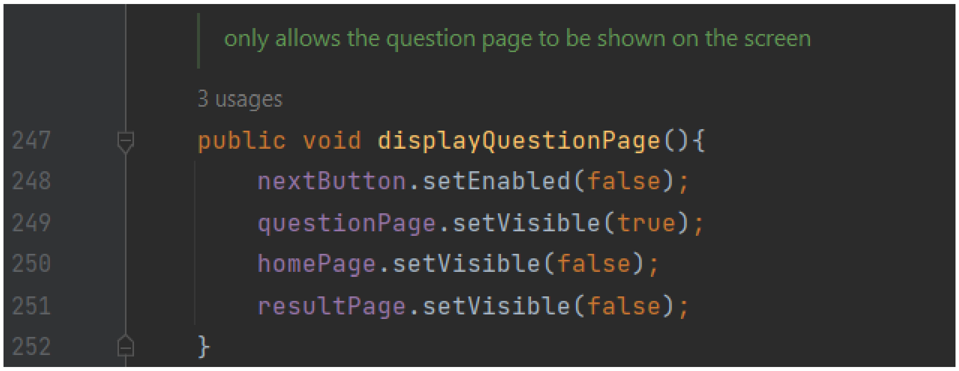
Furthermore, from lines 302 to 308 the program functions the same when the **JRadioButton** has been unselected. All other **JRadioButtons** follow the same logic.

The **ActionListener** classes are placed in the **controller** class because upon each interaction with the user, changes are made to the visual elements and algorithmic thinking is done. Moreover, making these classes private encapsulates the event-handling logic within the

controller, preventing unintentional modifications. This approach simplifies maintenance and enhances modularity by outlining responsibilities among the MVC components.

Displaying a certain page

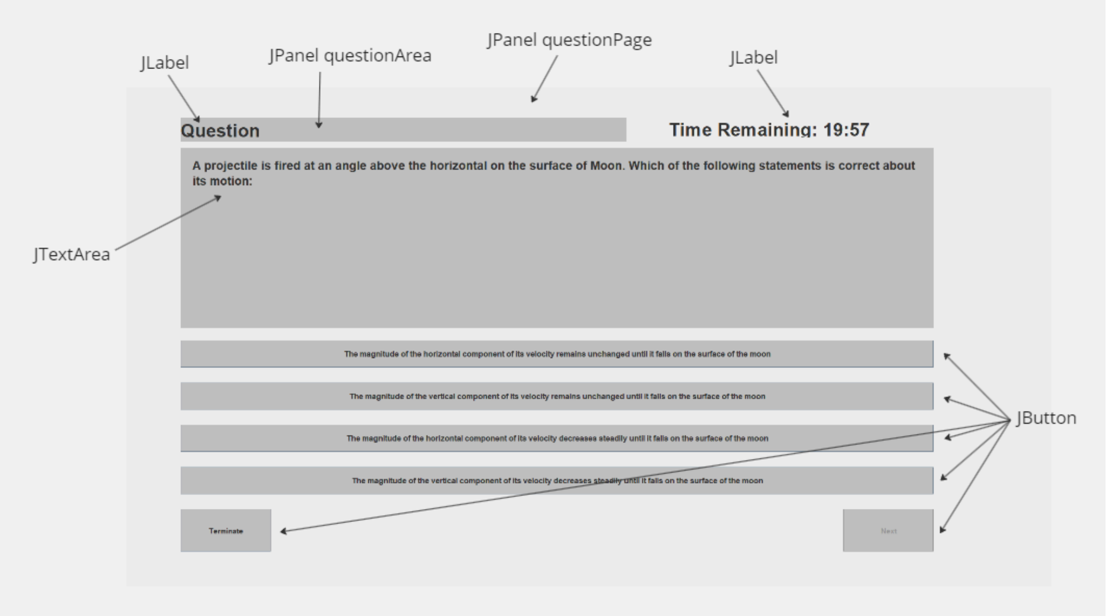
Document 11: Screenshot of the code of displaying the question page



```
only allows the question page to be shown on the screen
3 usages
public void displayQuestionPage(){
    nextButton.setEnabled(false);
    questionPage.setVisible(true);
    homePage.setVisible(false);
    resultPage.setVisible(false);
}
```

After the start button is pressed, this method is called. The method only allows the program to show the question page. Similar methods such as **displayHomePage** and **displayResultPage** follow the same logic. The next button is disabled first when the question is introduced because the user will only be able to click on it only if they get the correct answer to the question.

Document 12: Labeling the question page with the GUI components



A projectile is fired at an angle above the horizontal on the surface of Moon. Which of the following statements is correct about its motion:

The magnitude of the horizontal component of its velocity remains unchanged until it falls on the surface of the moon

The magnitude of the vertical component of its velocity remains unchanged until it falls on the surface of the moon

The magnitude of the horizontal component of its velocity decreases steadily until it falls on the surface of the moon

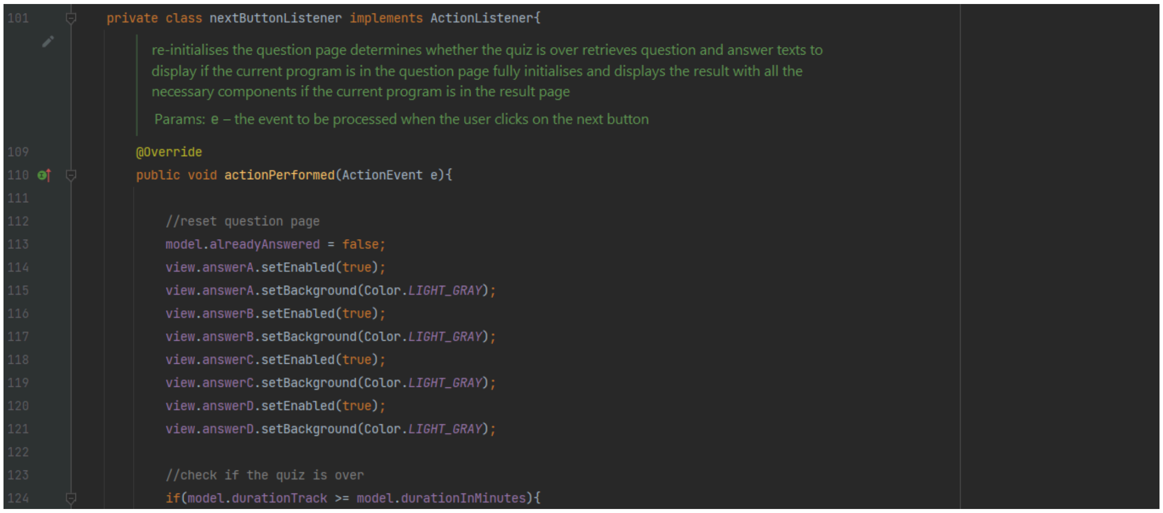
The magnitude of the vertical component of its velocity decreases steadily until it falls on the surface of the moon

7. **JTextArea**: used to display questions' text

Generating questions

The program generates questions sequentially, rather than all at once. This approach allows for evaluating the user's performance based on the number of questions completed within a selected time frame. It gives the user an idea of how many questions they can solve and their accuracy, just like exam conditions.

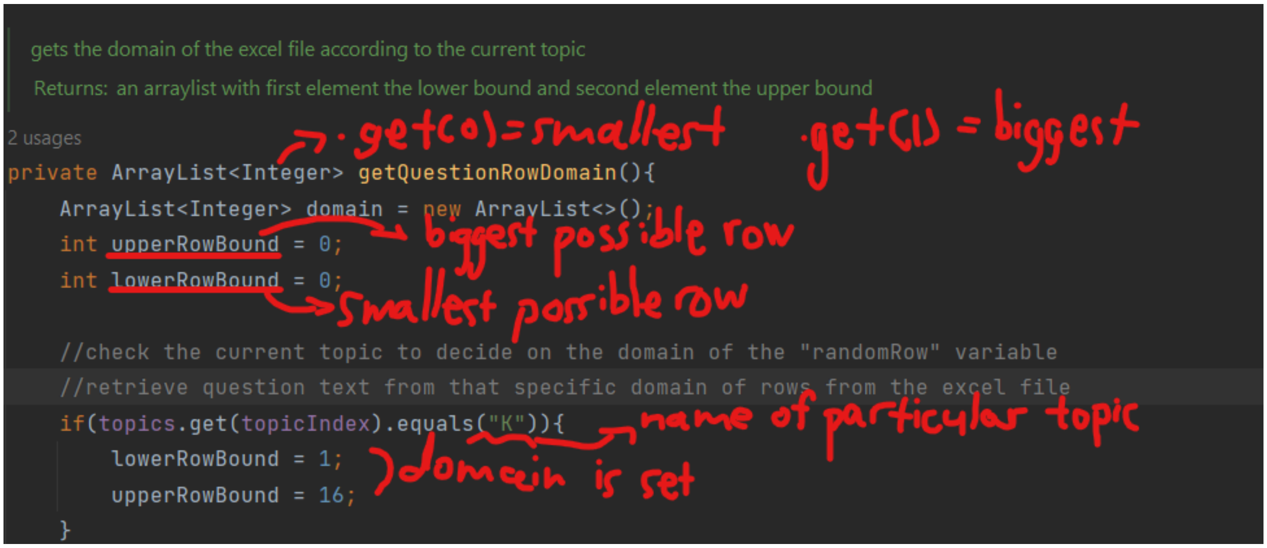
Document 13: Screenshot of the code of the class "nextButtonListener"



```
101    private class nextButtonListener implements ActionListener{
           re-initialises the question page determines whether the quiz is over retrieves question and answer texts to
           display if the current program is in the question page fully initialises and displays the result with all the
           necessary components if the current program is in the result page
           Params: e – the event to be processed when the user clicks on the next button
109        @Override
110        public void actionPerformed(ActionEvent e){
111
112            //reset question page
113            model.alreadyAnswered = false;
114            view.answerA.setEnabled(true);
115            view.answerA.setBackground(Color.LIGHT_GRAY);
116            view.answerB.setEnabled(true);
117            view.answerB.setBackground(Color.LIGHT_GRAY);
118            view.answerC.setEnabled(true);
119            view.answerC.setBackground(Color.LIGHT_GRAY);
120            view.answerD.setEnabled(true);
121            view.answerD.setBackground(Color.LIGHT_GRAY);
122
123            //check if the quiz is over
124            if(model.durationTrack >= model.durationInMinutes){
```

From lines 113 to 121, all the visual components' behaviors are reset to provide a new question page. At line 124, before generating the next question, the program checks if the user has time left to do another question by comparing **durationTrack** that represents the sum of the time all questions already done by the user are estimated to take, and **durationMinutes** the duration of quiz the user initially selected.

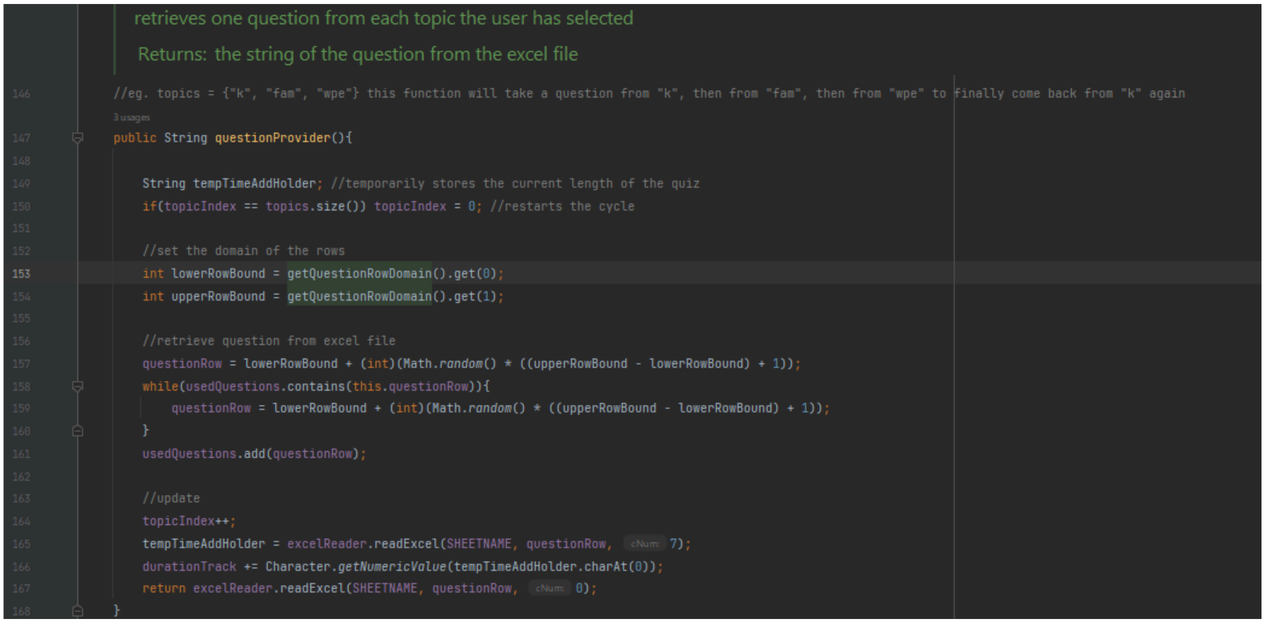
Document 14: Screenshot of the code of the method "getQuestionRowDomain"



```
gets the domain of the excel file according to the current topic
Returns: an arraylist with first element the lower bound and second element the upper bound
2 usages
private ArrayList<Integer> getQuestionRowDomain(){
    ArrayList<Integer> domain = new ArrayList<>();
    int upperRowBound = 0;
    int lowerRowBound = 0;

    //check the current topic to decide on the domain of the "randomRow" variable
    //retrieve question text from that specific domain of rows from the excel file
    if(topics.get(topicIndex).equals("K")){
        lowerRowBound = 1;
        upperRowBound = 16;
    }
```

In the case where another question can be generated, the domain of the row where to extract data from the Excel file is set. For instance, according to the Excel file, all questions from "kinematics" are from row 1 to 16. However, this method is not generalizable since the rows change when new questions are added in the Excel file.

Document 15: Screenshot of the code of the method "questionProvider"



```
        retrieves one question from each topic the user has selected
        Returns: the string of the question from the excel file
146     //eg. topics = {"k", "fam", "wpe"} this function will take a question from "k", then from "fam", then from "wpe" to finally come back from "k" again
        3 usages
147     public String questionProvider(){
148
149         String tempTimeAddHolder; //temporarily stores the current length of the quiz
150         if(topicIndex == topics.size()) topicIndex = 0; //restarts the cycle
151
152         //set the domain of the rows
153         int lowerRowBound = getQuestionRowDomain().get(0);
154         int upperRowBound = getQuestionRowDomain().get(1);
155
156         //retrieve question from excel file
157         questionRow = lowerRowBound + (int)(Math.random() * ((upperRowBound - lowerRowBound) + 1));
158         while(usedQuestions.contains(this.questionRow)){
159             questionRow = lowerRowBound + (int)(Math.random() * ((upperRowBound - lowerRowBound) + 1));
160         }
161         usedQuestions.add(questionRow);
162
163         //update
164         topicIndex++;
165         tempTimeAddHolder = excelReader.readExcel(SHEETNAME, questionRow, cNum: 7);
166         durationTrack += Character.getNumericValue(tempTimeAddHolder.charAt(0));
167         return excelReader.readExcel(SHEETNAME, questionRow, cNum: 0);
168     }
```

At line 150, if the program has generated at least one question from each topic the user has selected, then it resets to the first generated topic. This makes sure questions from every topic selected appear in the quiz. After, the program gets the domain of row at lines 153 and 154. Then, from lines 157 to 161, the program uses this domain to extract data from the Excel file using **Apache POI** and **java IO**:

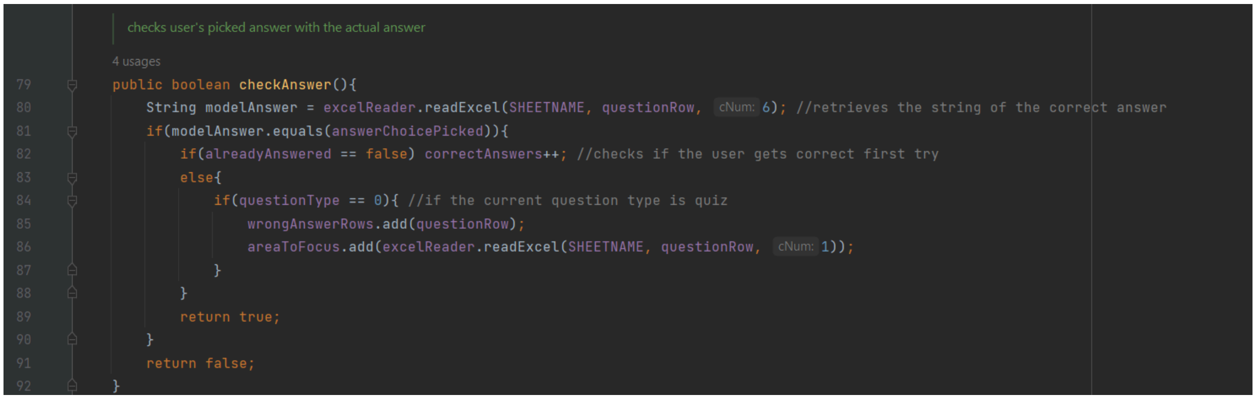
1. **poi.xssfworkbook**: used to represent the Excel file as a workbook that offers the ability to access specific cells. This was used to retrieve the question and answer texts from the Excel file.
2. **poi.sheet**: used to determine the sheet inside the Excel file to read
3. **poi.row**: used to determine the row of the data to read inside the sheet
4. **poi.cell**: used to determine the column of the data to read at previously defined row
5. **io.fileInputStream**: used to gain access to the Excel file.

The program ensures not providing the same question twice by storing the used questions in an **ArrayList** and checking if the new generated question is in that **ArrayList** from lines 158 to 161.

After generating each question, the estimated time to solve them is added to **durationTrack** as of lines 165 to 166.

Checking the user's answers

Document 16: Screenshot of the code of the method "checkAnswer"

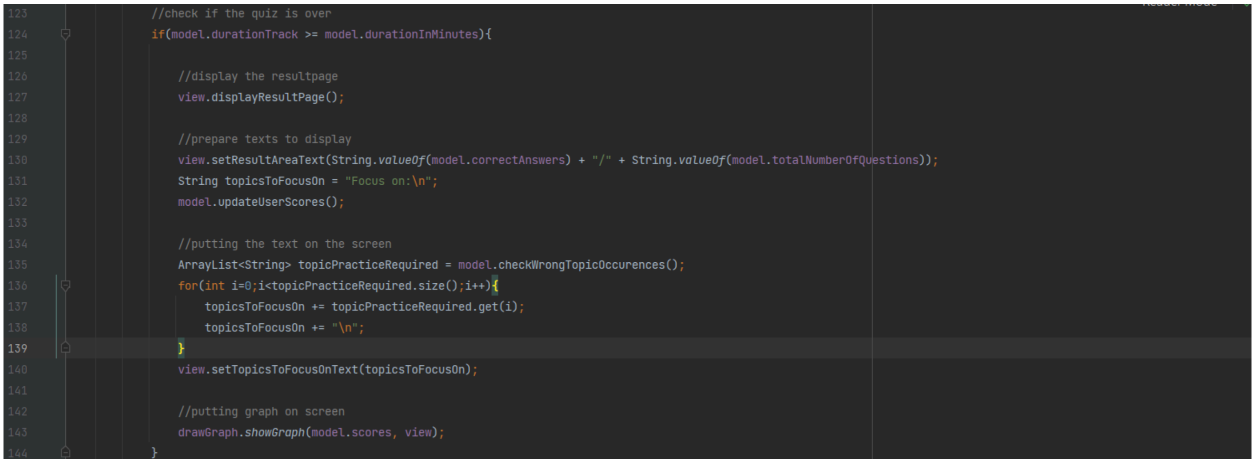


```
          checks user's picked answer with the actual answer

          4 usages
79        public boolean checkAnswer(){
80            String modelAnswer = excelReader.readExcel(SHEETNAME, questionRow,  cNum: 6); //retrieves the string of the correct answer
81            if(modelAnswer.equals(answerChoicePicked)){
82                if(alreadyAnswered == false) correctAnswers++; //checks if the user gets correct first try
83                else{
84                    if(questionType == 0){ //if the current question type is quiz
85                        wrongAnswerRows.add(questionRow);
86                        areaToFocus.add(excelReader.readExcel(SHEETNAME, questionRow,  cNum: 1));
87                    }
88                }
89                return true;
90            }
91            return false;
92        }
```

This method is called every time the user selects an answer choice. At line 80, the program retrieves the string of the correct answer from the Excel file. And from lines 81 to 92, the program compares the user’s answer choice to the correct answer to determine if the answer is correct. More specifically, at line 82, if the selected answer is the user’s first pick, then the total score is increased by 1. Otherwise, it means that the user has already chosen an incorrect answer before, therefore this question and its topic should be added to the **ArrayLists wrongAnswerRows** and **areaToFocus** that store all the incorrectly answered questions and their topics respectively. This allows the user to redo the questions in the practice questions again and facilitates the process of counting which topic the user made the most errors in by using the method *.frequency()*.

After returning true at line 89, the user is able to go to the next question.

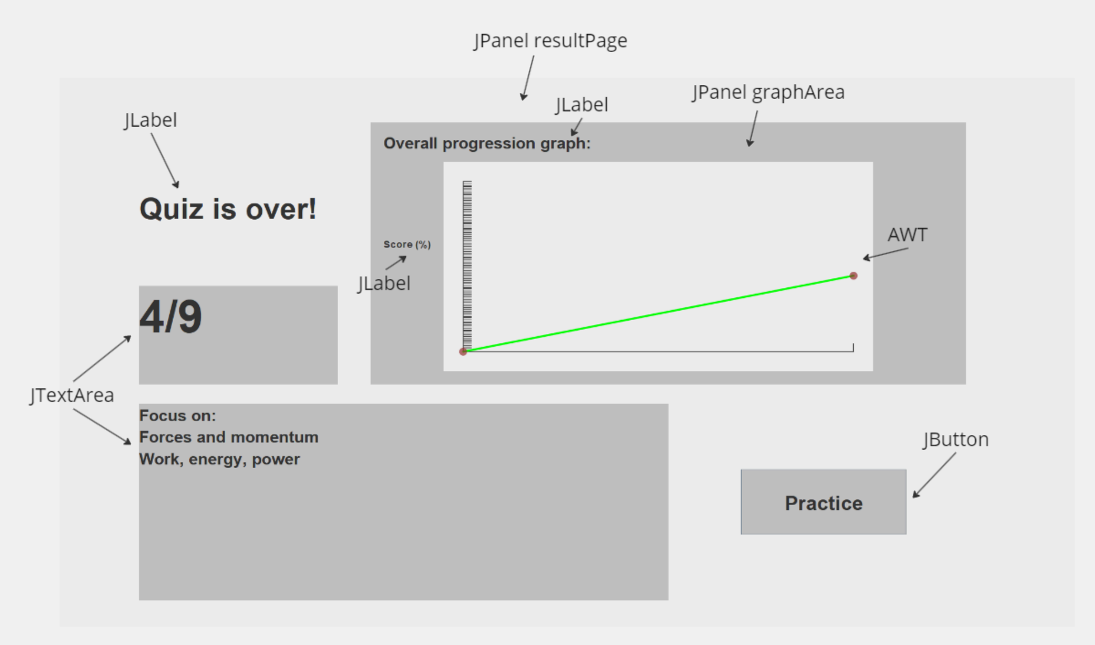
Document 17: Screenshot of the code of the class “nextButtonListener”



```
123         //check if the quiz is over
124         if(model.durationTrack >= model.durationInMinutes){
125 
126             //display the resultpage
127             view.displayResultPage();
128 
129             //prepare texts to display
130             view.setResultAreaText(String.valueOf(model.correctAnswers) + "/" + String.valueOf(model.totalNumberOfQuestions));
131             String topicsToFocusOn = "Focus on:\n";
132             model.updateUserScores();
133 
134             //putting the text on the screen
135             ArrayList<String> topicPracticeRequired = model.checkWrongTopicOccurences();
136             for(int i=0;i<topicPracticeRequired.size();i++){
137                 topicsToFocusOn += topicPracticeRequired.get(i);
138                 topicsToFocusOn += "\n";
139             }
140             view.setTopicsToFocusOnText(topicsToFocusOn);
141 
142             //putting graph on screen
143             drawGraph.showGraph(model.scores, view);
144         }
```

When **durationTrack** exceeds **quizDuration** after pressing the next button, the program switches to the **resultPage JPanel**. From line 138 to 143, all variables that associate with the necessary visual elements on the result page are updated and displayed.

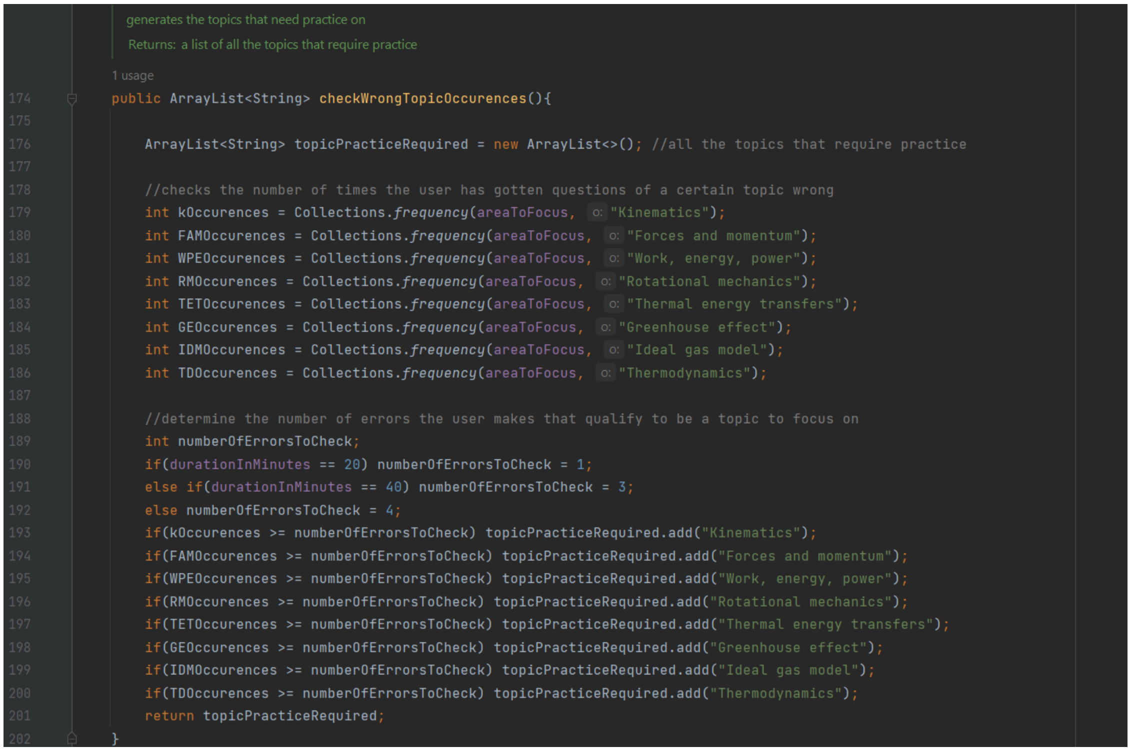
Document 18: Labeling the result page with the GUI components



8. **AWT**: Used to draw the graph

Evaluating user’s weak study areas

Document 17: Screenshot of the code of the method “checkWrongTopicOccurences”



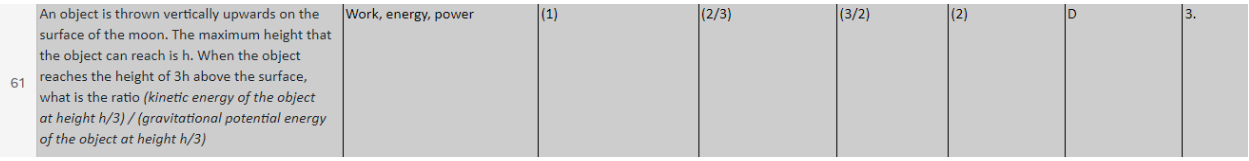
```
generates the topics that need practice on
Returns: a list of all the topics that require practice
1 usage
public ArrayList<String> checkWrongTopicOccurences(){

    ArrayList<String> topicPracticeRequired = new ArrayList<>(); //all the topics that require practice

    //checks the number of times the user has gotten questions of a certain topic wrong
    int kOccurences = Collections.frequency(areaToFocus,  o: "Kinematics");
    int FAMOccurences = Collections.frequency(areaToFocus,  o: "Forces and momentum");
    int WPEOccurences = Collections.frequency(areaToFocus,  o: "Work, energy, power");
    int RMOccurences = Collections.frequency(areaToFocus,  o: "Rotational mechanics");
    int TETOccurences = Collections.frequency(areaToFocus,  o: "Thermal energy transfers");
    int GEOccurences = Collections.frequency(areaToFocus,  o: "Greenhouse effect");
    int IDMOccurences = Collections.frequency(areaToFocus,  o: "Ideal gas model");
    int TDOccurences = Collections.frequency(areaToFocus,  o: "Thermodynamics");

    //determine the number of errors the user makes that qualify to be a topic to focus on
    int numberOfErrorsToCheck;
    if(durationInMinutes == 20) numberOfErrorsToCheck = 1;
    else if(durationInMinutes == 40) numberOfErrorsToCheck = 3;
    else numberOfErrorsToCheck = 4;
    if(kOccurences >= numberOfErrorsToCheck) topicPracticeRequired.add("Kinematics");
    if(FAMOccurences >= numberOfErrorsToCheck) topicPracticeRequired.add("Forces and momentum");
    if(WPEOccurences >= numberOfErrorsToCheck) topicPracticeRequired.add("Work, energy, power");
    if(RMOccurences >= numberOfErrorsToCheck) topicPracticeRequired.add("Rotational mechanics");
    if(TETOccurences >= numberOfErrorsToCheck) topicPracticeRequired.add("Thermal energy transfers");
    if(GEOccurences >= numberOfErrorsToCheck) topicPracticeRequired.add("Greenhouse effect");
    if(IDMOccurences >= numberOfErrorsToCheck) topicPracticeRequired.add("Ideal gas model");
    if(TDOccurences >= numberOfErrorsToCheck) topicPracticeRequired.add("Thermodynamics");
    return topicPracticeRequired;
}
```

From lines 179 to 186, the number of questions the user has gotten wrong in each topic is determined. The program in lines 189 to 200 evaluates if the user has made more mistakes in a specific topic than allowed according to the quiz duration selected.

1. 1 mistake allowed per topic in 20 mins
2. 3 mistake allowed per topic in 40 mins
3. 4 mistake allowed per topic in 60 mins

Based on this assessment, it classifies whether a topic requires focus or not.

Question pool



61	An object is thrown vertically upwards on the surface of the moon. The maximum height that the object can reach is h. When the object reaches the height of 3h above the surface, what is the ratio *(kinetic energy of the object at height h/3) / (gravitational potential energy of the object at height h/3)*	Work, energy, power	(1)	(2/3)	(3/2)	(2)	D	3.

Screenshot of a question set

It can be seen in the screenshot that I've added symbols such as **.** or **()** to cells that contained only numeric values, this is because the method *.getStringCellValue()* was used in the **ExcelReader** class. Therefore, the symbols modify the numeric values to string values for the program to retrieve data without error.

.getStringCellValue() only reads the string value of the cell

[1] "JDK 21 Documentation - Home." *Oracle Help Center*, docs.oracle.com/javase%2Ftutorial%2Fuiswing%2F%2F/events/actionlistener.html.

[2] Oracle. "Javax.swing (Java Platform SE 7)." *Docs.oracle.com*, docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html.

[3] "POI API Documentation." *Poi.apache.org*, poi.apache.org/apidocs/4.1/.

[4] "Busy Developers' Guide to HSSF and XSSF Features." *Poi.apache.org*, poi.apache.org/components/spreadsheet/quick-guide.html. Accessed 4 June 2024.

[5] "Drawing a Simple Line Graph in Java." *Stack Overflow*, stackoverflow.com/questions/8693342/drawing-a-simple-line-graph-in-java. Accessed 4 June 2024.

[6] "Your Home for IB Maths. Voted #1 IB Maths Resource in 2017!" *Revision Village - IB Maths*, 2017, www.revisionvillage.com/.