Appendix

First Interview with client /mr.x

Me: Hello sir, as we agreed, I will make a program based on your needs, could you tell me the problems you are currently facing?

Mr x: Hello, nice to meet you. I'm currently struggling with physics and my grades aren't too great. I was wondering if you could make a program to help me revise for the test. As you may know, most of the websites online require payments, and sometimes it is hard for me to find exercises to my needs.

Me: I see.

Mr x: So yeah, due to these factors, I cannot effectively manage my time and efficiency for my revision sessions. Sometimes it is also hard for me to keep track of my progressions, to know which topic I should aim to spend more time on.

Me: I understand, I think I have a solution to propose to you. What do you think about a program that will first give you a general quiz containing all the knowledge that's going to be in your exam. And after completing the quiz, the program will find your errors and weak areas to give you practice questions.

Mr x: I think that this will be helpful for my studies. Concerning some of the details, what will the home page look like?

Me: The home page will first show all the possible physics topics you've covered or you're going to cover, you will select the topics then the quiz will begin. Concerning the physics topics, I will need you to send me everything you're going to learn in your school year. I will try to make this process clear and easy for you.

Mr x: How long will the quiz be?

Me: This depends on you, do you have a preferable duration for the quiz?

Mr x: Yes, I’d say about 40 minutes. I like to feel challenged but if it gets too tedious, I usually lose focus.

Me: I see, would you want a function where you could choose the quiz length?

Mr x: Yes that would be perfect. Sometimes I might have a bit more time, and other times, I may need a quicker session.

Me: I understand, would you like any additional functionalities?

Mr x: Yes, I would like a function that allows me to track my progress. Something that records the results that I get on the tests with the dates that I took the test. I would also like this to be graphed with time on the x axis and my score on the y axis.

Me: Sounds good, do you want to include any other feature?

Mr. X: Yes, how about the feedback on the practice questions? How detailed will it be?

Me: The feedback would not be very detailed since it would require programming knowledge outside of my capacity. But after completing each practice question, the program will highlight the correct answer.

Mr. X: I see, say that I want to stop doing the practice questions, is there a way for me to stop doing them?

Me: Yes, you will be able to terminate the practice questions whenever you feel like it.

Mr. X: That flexibility sounds good. One last thing, will there be an option for customizing the difficulty level of the practice questions?

Me: Unfortunately I also think that this would take too much time on top of collecting practice questions, will it be okay if the questions are on a similar level of difficulty?

Mr. X: That’s not a problem. Thank you for addressing all my concerns. I'm looking forward to using this interface for my physics revision.

Me: You're welcome, Mr. X. I'm excited to work on this project with you, and I'm confident that this interface will greatly enhance your physics revision experience. If you have any further questions or suggestions as we progress, feel free to let me know.

Second interview with Mr.x:

Me: Hello, according to our last discussion, I would like to validate the features I plan on making.

Mr. X: Hello, sure.

Me: There will be 3 pages in total, one is the home page, one is the question page, and the last one is where you see your quiz results. In the home page, you will be able to select from 8 study topics and 3 quiz durations. I designed the program specifically so that you will only be able to start the quiz after you have selected at least 3 study topics, would that be fine for you?

Mr. X: Yes, everything seems good to me so far.

Me: Good to hear. Following what I said previously, once the quiz has started, the questions will be randomized based on the study topics you have chosen. For each question, you will select from 4 answer choices within which only 1 is correct. After selecting an answer, the answer choice will become red if it is incorrect and green vice versa, you will only be able to move to the next question once you’ve selected the correct answer. I have also made sure that you will not get the same question twice in the quiz.

Mr X: I see, everything is just as I expected. If I get a question wrong at first and correct afterwards, it doesn't count as a correct answer in the total score right?

Me: That is correct. During the quiz, you will have the ability to terminate it and the program will bring you back to the home page. A timer will also be available according to the duration you chose in the beginning, keep in mind that this will only show during the quiz and not in the practice questions since it might put some pressure on you.

Mr X: That sounds nice, I wouldn't have liked to be timed during the practice questions since its goal is for me to practice and to understand the concepts.

Me: Let's move on then. Once the quiz is over, on the result page, it will display the study topics you will need improvement on; it will display the score you got for the quiz; and it will also display a graph of your progression.

Mr X: Good, does the graph register the results I get with the dates in which I took them?

Me: Well, the progression graph can show you your progression only during program execution. Therefore, I'm afraid that this would not be possible.

Mr. X: Is it possible to store the results and make them appear the next time I open the program?

Me: Unfortunately, this would require much more time and work. I'm sorry that I cannot include this feature.

Mr X: It's fine, what happens after the quiz is over?

Me: After the quiz is over, you will be able to start doing practice questions on the study topics you selected. I've implemented a feature where the program lets you redo the questions where you have gotten wrong in the quiz.

Mr X: Ok! Perfect, everything seems good to me. Thank you.

Me: Thank you.

Final interview with Mr.X:

Me: Hello, we will go through the success criterias we've discussed previously in criterion A. Now that you have used this program for a week, would you be able to provide me feedback on each success criteria?

Mr.X: Hello, yes, I have familiarized myself with the program. I can tell you how I think about the success criterias.

Me: Perfect. To start with, do you think that the questions are randomized? In other words, did the order of the questions change each time you used the program?

Mr.X: Yes, I can't tell whether it is 'randomized' or not but the order of the questions felt very natural and seemed to have a nice flow. This was really helpful as it kept me focused for a longer period.

Me: Next, did you feel like the program was easy to navigate between the 3 pages in the program?

Mr.X: The program's layout and structure is rather simple, which makes it very easy to navigate. There are no potential distractions present on any of the pages, all of the information present in the program is relevant.

Me: Okay! In the home page, were you able to select study topics out of 8 and a quiz duration out of 3 with all the choices specified?

Mr.X: Yes, there are 8 different topics that I can choose from for the test. Regarding the possible durations, I was able to choose between 20, 40 and 60 minutes. I also am only able to start the quiz once I have selected at least three out of the 8 topics and a duration.

Me: Good to hear! Did you feel like the questions provided corresponded to the study topics you selected?

Mr.X: Yes! The questions that I encountered were very much about the three or more topics that I chose before starting the quiz. I did not encounter any questions which did not fall under one of my selected topics. For example, I did not get any questions about thermodynamics when I selected rotational mechanics.

Me: Yeah, it might be a bit hard for you to evaluate this. Now, was the timer only displaying and functioning during the quiz?

Mr.X: Indeed, I am only able to see the timer appear when I am actually taking the ***test,*** the timer does not appear when I am doing the practice questions or when I am in the home or result pages.

Me: Seems like you've understood how the program works! Just to make sure, were you only allowed to select among 4 answers for each question?

Mr.X: Yes, I was. There were 4 possible answers in each question.

Me: Were you able to terminate the quiz or the practice questions whenever you wanted?

Mr.X: Yes, there was a button in the bottom left of the screen where I could end the quiz at any time.

Me: I know this might be a bit hard for you to know, but do you think that all the answers given by the program were correct?

Mr.X: Yes, I do believe that the correct answers were indeed correct. Furthermore, the answers would turn green if I clicked on the correct one and red if I chose a wrong option, this made it clear when I was right or wrong. I can also only progress to the next question when I click on the right answer.

Me: Great! Did you encounter any repeating questions in the duration which you used the program?

Mr.X: No, I did not experience any such difficulties whilst using the program. The questions were unique each time I used the program.

Me: What did you think about the result page? Was your score correctly displayed? Do you think the program correctly evaluates the study topics you should focus on?

Mr.X: The result page was effective in telling me how well I did during the quiz and the ones preceding this quiz. My score was displayed in a fraction and corresponded every time to the score I got. Overall, the program does a good job at assessing my weak areas. I think that the number of incorrect answers here are appropriate to my test conditions.

Me: Was the progression graph fully functioning? Was it correctly plotting your score each time you took the quiz?

Mr.X: The progression graph was functioning perfectly well, it plotted my score on the graph with a dot and connected the dots with a straight line to provide a more visual representation of my progress. However, my scores were not saved after I closed and re-opened the program, which I thought would have been a useful tool to have.

Me: Right, unfortunately this was a feature I didn't have time to deal with, would you still be satisfied with the product?

Mr.X: Yes, the program still meets my expectations.

Me: Good! To finish with, were you able to redo the questions you answered incorrectly from the quiz in the practice questions?

Mr.X: Yes, the program allowed me to retake the questions that I got wrong in my first try.

Me: Great, do you think there could be any improvements?

Mr.X: Hmmm, yes, one of them is the format of the questions, sometimes it feels a bit repetitive. I would prefer if some questions included long answer response questions instead of multiple choice questions.

Me: I see, do you mean by having multiple question formats such as “true or false” or “enter the answer”?

Mr.X: Exactly.

Me: Do you think this program will help you in your physics revision sessions?

Mr.X: Yes of course. For now, the program has what I need and I believe that this program will help me better organize my time and revise with efficiency. It's a really nice tool to have for my test preparations!

Me: Perfect! Thank you for your feedback!

Mr.X: I'm overall very pleased with the product, see you!

Me: Thanks bye!

Code - Model Class

```
import java.awt.*;
import java.util.ArrayList;
import java.util.Collections;
```

```java
import java.util.List;

public class Model {


    //attributes
    ExcelReader excelReader = new ExcelReader();
    ArrayList<String> topics = new ArrayList<String>(); //stores the
topics selected by the user in the homepage
    ArrayList<Integer> usedQuestions = new ArrayList<Integer>();
//stores the questions that the user has already answered
    ArrayList<Integer> wrongAnswerRows = new ArrayList<Integer>();
//stores the row of the questions from the excel file that the user
ansered incorrectly
    ArrayList<String> areaToFocus = new ArrayList<>(); //stores the
topics the user should focus on
    int questionRow; //row of the question from the excel file that
will be retrieved by the excel reader
    int correctAnswers = 0;
    String answerChoicePicked = ""; //stores the string of the answer
that the user has picked
    boolean alreadyAnswered = false; //value to check if the user has
already answered a question
    int durationInMinutes; //length of the quiz
    int durationTrack = 0; //length in the current program execution
    int topicIndex = 0; //iterator used to retrieve question from the
excel file
    boolean isTopicsSelected = false; //value to check if at least 3
topics are selected
    boolean isDurationSelected = false; //value to check if a quiz
duration is selected
    int totalNumberOfQuestions = 1;
    int wrongQuestionCount = 0; //a counter to check if all the
initial wrongly answered questions has been answered during practice
```

```
questions / also used as index of rows of the wrongly answered
questions
    byte questionType = 0; //0 is quiz, 1 is practice questions
    final String SHEETNAME = "Feuil1";
    List<Integer> scores = new ArrayList<Integer>(){{
        add(0); //initialises the origin point of the graph
    }};


    //constructor
    Model(ExcelReader excelReader){
        excelReader = this.excelReader;
    }


    //methods

    //retrieve texts of the answer choices
    /**
     * gets the texts of answer A from excel file
     * @return the text retrieved
     */
    public String retrieveAnswerAText(){
        return excelReader.readExcel(SHEETNAME, questionRow, 2);
    }

    /**
     * gets the texts of answer B from excel file
     * @return the text retrieved
     */
    public String retrieveAnswerBText(){
        return excelReader.readExcel(SHEETNAME, questionRow, 3);
    }

    /**
```

```
     * gets the texts of answer C from excel file
     * @return the text retrieved
     */
   public String retrieveAnswerCText(){
       return excelReader.readExcel(SHEETNAME, questionRow, 4);
   }

    /**
     * gets the texts of answer D from excel file
     * @return the text retrieved
     */
   public String retrieveAnswerDText(){
       return excelReader.readExcel(SHEETNAME, questionRow, 5);
   }


    /**
     * checks user's picked answer with the actual answer
     * @param view an instance of the class 'view' in order to modify
it
     */
   public boolean checkAnswer(){
       String modelAnswer = excelReader.readExcel(SHEETNAME,
questionRow, 6); //retrieves the string of the correct answer
       if(modelAnswer.equals(answerChoicePicked)){
           if(alreadyAnswered == false) correctAnswers++; //checks if
the user gets correct first try
           else{
               if(questionType == 0){ //if the current question type
is quiz
                   wrongAnswerRows.add(questionRow);
                   areaToFocus.add(excelReader.readExcel(SHEETNAME,
questionRow, 1));
               }
           }
```

```
                return true;
            }
        return false;
    }

    /**
     * gets the domain of the excel file according to the current topic
     * @return an arraylist with first element the lower bound and
second element the upper bound
     */
    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;
        }
        else if(topics.get(topicIndex).equals("FAM")){
            lowerRowBound = 17;
            upperRowBound = 46;
        }
        else if(topics.get(topicIndex).equals("WPE")){
            lowerRowBound = 47;
            upperRowBound = 67;
        }
        else if(topics.get(topicIndex).equals("RM")){
            lowerRowBound = 68;
            upperRowBound = 75;
        }
        else if(topics.get(topicIndex).equals("TET")){
            lowerRowBound = 76;
            upperRowBound = 90;
        }
        else if(topics.get(topicIndex).equals("GE")){
            lowerRowBound = 91;
            upperRowBound = 101;
        }
        else if(topics.get(topicIndex).equals("IGM")){
            lowerRowBound = 102;
            upperRowBound = 113;
        }
        else if(topics.get(topicIndex).equals("TD")){
            lowerRowBound = 114;
            upperRowBound = 119;
        }
        domain.add(lowerRowBound);
        domain.add(upperRowBound);
        return domain;
    }

    /**
     * retrieves one question from each topic the user has selected
     * @return the string of the question from the excel file
     */
    //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
    public String questionProvider(){

        String tempTimeAddHolder; //temporarily stores the current
length of the quiz
        if(topicIndex == topics.size()) topicIndex = 0; //restarts the
cycle
```

```
        //set the domain of the rows
        int lowerRowBound = getQuestionRowDomain().get(0);
        int upperRowBound = getQuestionRowDomain().get(1);

        //retrieve question from excel file
        questionRow = lowerRowBound + (int)(Math.random() *
((upperRowBound - lowerRowBound) + 1));
        while(usedQuestions.contains(this.questionRow)){
            questionRow = lowerRowBound + (int)(Math.random() *
((upperRowBound - lowerRowBound) + 1));
        }
        usedQuestions.add(questionRow);

        //update
        topicIndex++;
        tempTimeAddHolder = excelReader.readExcel(SHEETNAME,
questionRow, 7);
        durationTrack +=
Character.getNumericValue(tempTimeAddHolder.charAt(0));
        return excelReader.readExcel(SHEETNAME, questionRow, 0);
    }

    /**
     * generates the topics that need practice on
     * @return a list of all the topics that require practice
     */
    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,
"Kinematics");
```

```
        int FAMOccurences = Collections.frequency(areaToFocus, "Forces
and momentum");
        int WPEOccurences = Collections.frequency(areaToFocus, "Work,
energy, power");
        int RMOccurences = Collections.frequency(areaToFocus,
"Rotational mechanics");
        int TETOccurences = Collections.frequency(areaToFocus,
"Thermal energy transfers");
        int GEOccurences = Collections.frequency(areaToFocus,
"Greenhouse effect");
        int IDMOccurences = Collections.frequency(areaToFocus, "Ideal
gas model");
        int TDOccurences = Collections.frequency(areaToFocus,
"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;
    }

    /**
     * updates the scores the user got from all the quizzes
     */
    public void updateUserScores(){
        int scoreToAdd = (int)(((double) correctAnswers / (double)
totalNumberOfQuestions) * 100);
        scores.add(scoreToAdd);
    }
}
```

Code - View Class

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class View extends JFrame{

    //attributes
    JButton startQuizButton = new JButton("Start quiz");
    private final JButton terminateButton = new JButton("Terminate");
    JButton nextButton = new JButton("Next");
    JButton startPracticeQuestionsButton = new JButton("Practice");
    private JLabel topicSelection = new JLabel("Select your topics:
(atleast 3)");
    private JLabel preferredLength = new JLabel("Preferred length
(mins): (need to select)");
    private JLabel quizOver = new JLabel("Quiz is over!");
    private JLabel title = new JLabel("Better Physics");
    private JLabel questionTextOnTopPage = new JLabel("Question");
    private JLabel graphAreaText = new JLabel("Overall progression
graph:");
    private JLabel graphYAxisLabel = new JLabel("Score (%)");
    private JTextArea questionTextArea = new JTextArea("");
    private JTextArea resultTextArea = new JTextArea("0/0");
    private JTextArea topicsToFocusOn = new JTextArea("Focus
On:\nNothing! You've mastered these topics!");
    String[] possibleQuizDurations = {"20", "40", "60"};
    JComboBox<String> quizDuration = new
JComboBox(possibleQuizDurations);
    JRadioButton kinematics = new JRadioButton("Kinematics");
    JRadioButton forcesAndMomentum = new JRadioButton("Forces and
Momentum");
    JRadioButton workPowerEnergy = new JRadioButton("Work, power,
energy");
    JRadioButton idealGasModel = new JRadioButton("Ideal gas model");
    JRadioButton greenhouseEffect = new JRadioButton("Greenhouse
effect");
    JRadioButton thermalDynamics = new
JRadioButton("Thermaldynamics");
    JRadioButton rotationalMechanics = new JRadioButton("Rotational
mechanics");
    JRadioButton thermalEnergyTransfer = new JRadioButton("Thermal
energy transfer");
    JButton answerA = new JButton("A");
    JButton answerB = new JButton("B");
    JButton answerC = new JButton("C");
    JButton answerD = new JButton("D");
    JPanel homePage = new JPanel();
    JPanel questionPage = new JPanel();
    JPanel resultPage = new JPanel();


    //Constructor
    public View(){
        //initialise settings for the screen
        setTitle("Better physics");
        setLayout(null);
        setSize(1440, 1024);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        //JPANELS
        homePage.setBounds(0, 0, 1440, 1024);
        homePage.setLayout(null);
        questionPage.setBounds(0, 0, 1440, 1024);
        questionPage.setLayout(null);
        JPanel topics = new JPanel();
        topics.setBounds(130, 200, 400, 520);
        topics.setLayout(null);
        topics.setBackground(Color.LIGHT_GRAY);
        JPanel questionArea = new JPanel();
        questionArea.setBounds(90, 100, 1250, 300);
        questionArea.setLayout(null);
        questionArea.setBackground(Color.LIGHT_GRAY);
        resultPage.setBounds(0, 0, 1440, 1024);
        resultPage.setLayout(null);

        JPanel graphArea = new JPanel();
        graphArea.setBounds(470, 70, 900, 400);
        graphArea.setLayout(null);
        graphArea.setBackground(Color.LIGHT_GRAY);


        //TEXTS AND LABELS
        title.setBounds(800, 50, 304, 222);
        title.setFont(new Font(null, Font.BOLD, 43));
        topicSelection.setBounds(140, 140, 350, 43);
        topicSelection.setFont(new Font(null, Font.BOLD, 24));
        preferredLength.setBounds(830, 450, 400, 50);
        preferredLength.setFont(new Font(null, Font.BOLD, 20));
        questionTextOnTopPage.setBounds(90, 50, 740, 40);
        questionTextOnTopPage.setFont(new Font(null, Font.BOLD, 30));
        questionTextOnTopPage.setBackground(Color.LIGHT_GRAY);
        questionTextOnTopPage.setOpaque(true);
        questionTextArea.setBounds(110, 115, 1210, 260);
        questionTextArea.setFont(new Font(null, Font.BOLD, 20));
        questionTextArea.setBackground(Color.LIGHT_GRAY);
        questionTextArea.setWrapStyleWord(true);
        questionTextArea.setEditable(false);
        questionTextArea.setLineWrap(true);
        quizOver.setBounds(120, 100, 300, 200);
        quizOver.setFont(new Font(null, Font.BOLD, 45));
        resultTextArea.setBounds(120, 320, 300, 150);
        resultTextArea.setFont(new Font(null, Font.BOLD, 70));
        resultTextArea.setBackground(Color.LIGHT_GRAY);
        resultTextArea.setWrapStyleWord(true);
        resultTextArea.setLineWrap(true);
        resultTextArea.setEditable(false);
        graphAreaText.setBounds(20, 15, 400, 30);
        graphAreaText.setFont(new Font(null, Font.BOLD, 25));
        graphYAxisLabel.setBounds(20, 160, 80, 50);
        graphYAxisLabel.setFont(new Font(null, Font.BOLD, 15));
        topicsToFocusOn.setBounds(120, 500, 800, 300);
        topicsToFocusOn.setFont(new Font(null, Font.BOLD, 25));
        topicsToFocusOn.setBackground(Color.LIGHT_GRAY);
        topicsToFocusOn.setWrapStyleWord(true);
        topicsToFocusOn.setLineWrap(true);
        topicsToFocusOn.setEditable(false);


        //BUTTONS
        startQuizButton.setBounds(830, 300, 250, 80);
        startQuizButton.setFont(new Font(null, Font.BOLD, 30));
        startQuizButton.setBackground(Color.GRAY);
        startQuizButton.setEnabled(false);
        answerA.setBounds(90, 420, 1250, 45);
        answerA.setBackground(Color.LIGHT_GRAY);
        answerA.setFont(new Font(null, Font.BOLD, 13));
        answerB.setBounds(90, 490, 1250, 45);
        answerB.setBackground(Color.LIGHT_GRAY);
        answerB.setFont(new Font(null, Font.BOLD, 13));
        answerC.setBounds(90, 560, 1250, 45);
        answerC.setBackground(Color.LIGHT_GRAY);
        answerC.setFont(new Font(null, Font.BOLD, 13));
        answerD.setBounds(90, 630, 1250, 45);
        answerD.setBackground(Color.LIGHT_GRAY);
        answerD.setFont(new Font(null, Font.BOLD, 13));
        terminateButton.setBounds(90, 700, 150, 70);
        terminateButton.setBackground(Color.LIGHT_GRAY);
        nextButton.setBounds(1190, 700, 150, 70);
        nextButton.setBackground(Color.LIGHT_GRAY);
        nextButton.setEnabled(false);
        startPracticeQuestionsButton.setBounds(1030, 600, 250, 100);
        startPracticeQuestionsButton.setBackground(Color.LIGHT_GRAY);
        startPracticeQuestionsButton.setFont(new Font(null, Font.BOLD,
30));


        //COMBOBOXES
        quizDuration.setBounds(830, 500, 150, 30);


        //'topics' PANEL
        kinematics.setBounds(20, 30, 170, 90);
        forcesAndMomentum.setBounds(210, 30, 170, 90);
        workPowerEnergy.setBounds(20, 150, 170, 90);
        idealGasModel.setBounds(210, 150, 170, 90);
        greenhouseEffect.setBounds(20, 270, 170, 90);
        thermalDynamics.setBounds(210, 270, 170, 90);
        rotationalMechanics.setBounds(20, 390, 170, 90);
        thermalEnergyTransfer.setBounds(210, 390, 170, 90);
        topics.add(kinematics);
        topics.add(forcesAndMomentum);
        topics.add(workPowerEnergy);
        topics.add(idealGasModel);
        topics.add(greenhouseEffect);
        topics.add(thermalDynamics);
        topics.add(rotationalMechanics);
        topics.add(thermalEnergyTransfer);


        //'graphArea' PANEL
        graphArea.add(graphAreaText);
        graphArea.add(graphYAxisLabel);


        //'questionArea' PANEL
        questionArea.add(questionTextOnTopPage);
        questionArea.add(questionTextArea);


        //HOME PAGE
        homePage.add(title);
        homePage.add(topicSelection);
        homePage.add(quizDuration);
        homePage.add(preferredLength);
        homePage.add(startQuizButton);
        homePage.add(topics);


        //QUESTION PAGE
        questionPage.add(questionArea);
        questionPage.setComponentZOrder(questionTextOnTopPage, 0);
        questionPage.add(answerA);
        questionPage.add(answerB);
        questionPage.add(answerC);
        questionPage.add(answerD);
        questionPage.add(questionTextArea);
        questionPage.setComponentZOrder(questionTextArea, 0);
        questionPage.add(terminateButton);
        questionPage.add(nextButton);


        //RESULT PAGE
        resultPage.add(quizOver);
        resultPage.add(resultTextArea);
        resultPage.add(graphArea);
        resultPage.add(topicsToFocusOn);
        resultPage.add(startPracticeQuestionsButton);


        //JFRAME
        add(homePage);
        add(questionPage);
        add(resultPage);
        //only the main page should be visible when the program first
executes
        questionPage.setVisible(false);
        resultPage.setVisible(false);
    }


    //Methods

    //Button listeners
    public void addStartButtonListener(ActionListener listener)
{startQuizButton.addActionListener(listener);}
```

```
    public void addNextButtonListener(ActionListener listener)
{nextButton.addActionListener(listener);}
    public void addStartPracticeQuestionsButtonListener(ActionListener
listener) {startPracticeQuestionsButton.addActionListener(listener);}
    public void addTerminateButtonListener(ActionListener listener)
{terminateButton.addActionListener(listener);}
    public void addAnswerAButtonListener(ActionListener listener)
{answerA.addActionListener(listener);}
    public void addAnswerBButtonListener(ActionListener listener)
{answerB.addActionListener(listener);}
    public void addAnswerCButtonListener(ActionListener listener)
{answerC.addActionListener(listener);}
    public void addAnswerDButtonListener(ActionListener listener)
{answerD.addActionListener(listener);}
    public void addQuizDurationDropBoxListener(ActionListener
listener) {quizDuration.addActionListener(listener);}
    public void addKinematicsListener(ActionListener listener)
{kinematics.addActionListener(listener);}
    public void addForcesAndMomentumListener(ActionListener listener)
{forcesAndMomentum.addActionListener(listener);}
    public void addWorkPowerEnergyListener(ActionListener listener)
{workPowerEnergy.addActionListener(listener);}
    public void addRotationalMechanicsListener(ActionListener
listener) {rotationalMechanics.addActionListener(listener);}
    public void addThermalEnergyTransfersListener(ActionListener
listener) {thermalEnergyTransfer.addActionListener(listener);}
    public void addGreenhouseEffectListener(ActionListener listener)
{greenhouseEffect.addActionListener(listener);}
    public void addIdealGasLawsListener(ActionListener listener)
{idealGasModel.addActionListener(listener);}
    public void addThermodynamicsListener(ActionListener listener)
{thermalDynamics.addActionListener(listener);}


    //display pages
    /**
     * only allows the home page to be shown on the screen
     */
    public void displayHomePage(){
        homePage.setVisible(true);
        questionPage.setVisible(false);
        resultPage.setVisible(false);
        startQuizButton.setEnabled(false);
    }

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

    /**
     * only allows the result page to be shown on the screen
     */
    public void displayResultPage(){
        resultPage.setVisible(true);
        homePage.setVisible(false);
        questionPage.setVisible(false);
    }


    //Mutators
    public void setQuestionTextArea(String
questionText){questionTextArea.setText(questionText);}
```

```
    public void setAnswerAText(String
answerAText){answerA.setText(answerAText);}
    public void setAnswerBText(String
answerBText){answerB.setText(answerBText);}
    public void setAnswerCText(String
answerCText){answerC.setText(answerCText);}
    public void setAnswerDText(String
answerDText){answerD.setText(answerDText);}
    public void setResultAreaText(String
score){resultTextArea.setText(score);}
    public void setTopicsToFocusOnText(String
focus){topicsToFocusOn.setText(focus);}

    //change button behaviors

    /**
     * change buttons behaviors when A is picked
     */
    public void answerChoiceAPicked(){
        answerA.setBackground(Color.green);
        answerB.setEnabled(false);
        answerC.setEnabled(false);
        answerD.setEnabled(false);
    }
    /**
     * change buttons behaviors when B is picked
     */
    public void answerChoiceBPicked(){
        answerB.setBackground(Color.green);
        answerA.setEnabled(false);
        answerC.setEnabled(false);
        answerD.setEnabled(false);
    }
    /**
     * change buttons behaviors when C is picked
     */
    public void answerChoiceCPicked(){
        answerC.setBackground(Color.green);
        answerB.setEnabled(false);
        answerA.setEnabled(false);
        answerD.setEnabled(false);
    }
    /**
     * change buttons behaviors when D is picked
     */
    public void answerChoiceDPicked(){
        answerD.setBackground(Color.green);
        answerB.setEnabled(false);
        answerC.setEnabled(false);
        answerA.setEnabled(false);
    }
}
```

Code - Controller Class

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

public class Controller {

    //attributes
    private Model model;
    private View view;
    private ExcelReader excelReader;
    private CountdownTimer countdownTimer;
    private DrawGraph drawGraph;



    //MAIN PROGRAM EXECUTION
    public static void main(String[] args){

        View view = new View();
        ExcelReader excelReader = new ExcelReader();
        Model model = new Model(excelReader);
        DrawGraph drawGraph = new DrawGraph(model.scores);
        CountdownTimer countDownTimer = new CountdownTimer(model,
view, drawGraph);

        view.setVisible(true);

        new Controller(model, view, excelReader, countDownTimer,
drawGraph);
    }


    //constructor
    public Controller(Model model, View view, ExcelReader excelReader,
CountdownTimer countdownTimer, DrawGraph drawGraph){

        //construct the classes
        this.model = model;
        this.view = view;
        this.excelReader = excelReader;
        this.countdownTimer = countdownTimer;
        this.drawGraph = drawGraph;

        //construct the listeners
        this.view.addStartButtonListener(new
startQuizButtonListener());
        this.view.addNextButtonListener(new nextButtonListener());
        this.view.addTerminateButtonListener(new
terminateButtonListener());
        this.view.addStartPracticeQuestionsButtonListener(new
startPracticeQuestionsListener());
```

```
        this.view.addAnswerAButtonListener(new
answerAButtonListener());
        this.view.addAnswerBButtonListener(new
answerBButtonListener());
        this.view.addAnswerCButtonListener(new
answerCButtonListener());
        this.view.addAnswerDButtonListener(new
answerDButtonListener());
        this.view.addQuizDurationDropBoxListener(new
quizDurationDropBoxListener());
        this.view.addKinematicsListener(new kinematicsListener());
        this.view.addForcesAndMomentumListener(new
forcesAndMomentumListener());
        this.view.addWorkPowerEnergyListener(new
workPowerEnergyListener());
        this.view.addRotationalMechanicsListener(new
rotationalMechanicsListener());
        this.view.addThermalEnergyTransfersListener(new
thermalEnergyTransferListener());
        this.view.addGreenhouseEffectListener(new
greenhouseEffectListener());
        this.view.addIdealGasLawsListener(new
idealGasModelListener());
        this.view.addThermodynamicsListener(new
thermalDynamicsListener());
    }


    //Classes for listener
    private class startQuizButtonListener implements ActionListener{

        /**
         * fully initialises and displays the question page with all
the necessary components
```

```
         * @param e the event to be processed when the user clicks on
the start button
         */
        @Override
        public void actionPerformed(ActionEvent e){

            //initialise the screen and variables
            view.displayQuestionPage();
            model.correctAnswers = 0;
            model.questionType = 0;
            countdownTimer.startTimer();
            String questionText;

            //retrieve question and answer texts
            questionText = model.questionProvider();
            String answerAText = model.retrieveAnswerAText();
            String answerBText = model.retrieveAnswerBText();
            String answerCText = model.retrieveAnswerCText();
            String answerDText = model.retrieveAnswerDText();

            //display texts retrieved on the screen
            view.setQuestionTextArea(questionText);
            view.setAnswerAText(answerAText);
            view.setAnswerBText(answerBText);
            view.setAnswerCText(answerCText);
            view.setAnswerDText(answerDText);

            //add the timer to the screen
            countdownTimer.timerLabel.setText("00:00");
            countdownTimer.timerLabel.setFont(new Font(null,
Font.BOLD, 30));
            countdownTimer.timerLabel.setVisible(true);
            view.questionPage.add(countdownTimer.timerLabel);
        }
    }
```

```
   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
          * @param e the event to be processed when the user clicks on
the next button
          */
         @Override
         public void actionPerformed(ActionEvent e){

             //reset question page
             model.alreadyAnswered = false;
             view.answerA.setEnabled(true);
             view.answerA.setBackground(Color.LIGHT_GRAY);
             view.answerB.setEnabled(true);
             view.answerB.setBackground(Color.LIGHT_GRAY);
             view.answerC.setEnabled(true);
             view.answerC.setBackground(Color.LIGHT_GRAY);
             view.answerD.setEnabled(true);
             view.answerD.setBackground(Color.LIGHT_GRAY);

             //check if the quiz is over
             if(model.durationTrack >= model.durationInMinutes){

                 //display the resultpage
                 view.displayResultPage();

                 //prepare texts to display
```

```
view.setResultAreaText(String.valueOf(model.correctAnswers) + "/" +
String.valueOf(model.totalNumberOfQuestions));
                    String topicsToFocusOn = "Focus on:\n";
                    model.updateUserScores();

                    //putting the text on the screen
                    ArrayList<String> topicPracticeRequired =
model.checkWrongTopicOccurences();
                    for(int i=0;i<topicPracticeRequired.size();i++){
                        topicsToFocusOn += topicPracticeRequired.get(i);
                        topicsToFocusOn += "\n";
                    }
                    view.setTopicsToFocusOnText(topicsToFocusOn);

                    //putting graph on screen
                    drawGraph.showGraph(model.scores, view);
                }

                //if the quiz is not over
                else{

                    //prepare page and variables for the next question
                    view.displayQuestionPage();
                    model.totalNumberOfQuestions++;
                    String questionText;

                    //if the current type is quiz OR all the wrongly
answered questions has been ansered again in the practice questions
                    if(model.questionType == 0 || model.wrongQuestionCount
== model.wrongAnswerRows.size())
                        questionText = model.questionProvider();
//retrieve random question
                    else{
```

```
                              //feature to redo the question gotten wrong in
practice questions again
                              model.questionRow =
model.wrongAnswerRows.get(model.wrongQuestionCount);
                              questionText =
excelReader.readExcel(model.SHEETNAME, model.questionRow, 0);
                              model.wrongQuestionCount++;
                        }

                        //retrieve answer texts
                        String answerAText = model.retrieveAnswerAText();
                        String answerBText = model.retrieveAnswerBText();
                        String answerCText = model.retrieveAnswerCText();
                        String answerDText = model.retrieveAnswerDText();

                        //display the texts
                        view.setQuestionTextArea(questionText);
                        view.setAnswerAText(answerAText);
                        view.setAnswerBText(answerBText);
                        view.setAnswerCText(answerCText);
                        view.setAnswerDText(answerDText);
                        view.nextButton.setEnabled(false); //disable the
button
                  }
            }
      }

    private class terminateButtonListener implements ActionListener{
           /**
            * resets all the variables and brings the user back to the
home page
            * @param e the event to be processed when the user clicks on
the terminate button
            */
           @Override
```

```
        public void actionPerformed(ActionEvent e){

            //reset everything
            countdownTimer.timer.stop();
            model.durationTrack = 0;
            model.isDurationSelected = false;
            model.usedQuestions.clear();
            model.correctAnswers = 0;
            model.wrongQuestionCount = 0;
            model.wrongAnswerRows.clear();
            model.totalNumberOfQuestions = 1;
            model.areaToFocus.clear();
            model.topicIndex = 0;
            model.alreadyAnswered = false;
            view.answerA.setEnabled(true);
            view.answerA.setBackground(Color.LIGHT_GRAY);
            view.answerB.setEnabled(true);
            view.answerB.setBackground(Color.LIGHT_GRAY);
            view.answerC.setEnabled(true);
            view.answerC.setBackground(Color.LIGHT_GRAY);
            view.answerD.setEnabled(true);
            view.answerD.setBackground(Color.LIGHT_GRAY);

            //display homepage
            view.displayHomePage();
        }
    }

   private class startPracticeQuestionsListener implements
ActionListener{
       /**
        * starts the practice questions
        * retrieves the question and answer texts from the excel file
to display
```

```
        * @param e the event to be processed when the user clicks on
the practice button
        */
       @Override
       public void actionPerformed(ActionEvent e)  {
           if(e.getSource() == view.startPracticeQuestionsButton){

               //initalise and prepare variables
               model.questionType = 1;
               view.displayQuestionPage();

               //this exceeds the total number of minutes added from
my question pool so make sure the practice questions will keep
running
               model.durationInMinutes = 1000;

               //make sure the timer doesnt show and run
               countdownTimer.timer.stop();
               countdownTimer.timerLabel.setVisible(false);

               //checks if there are any questions the user has
gotten wrong
               String questionText;
               if(model.wrongQuestionCount ==
model.wrongAnswerRows.size()){
                   //retrieve random question according to the topics
the user has selected in the beginning
                   questionText = model.questionProvider();
               }
               else{
                   //redo the wrongly answered questions
                   model.questionRow =
model.wrongAnswerRows.get(model.wrongQuestionCount);
                   questionText =
excelReader.readExcel(model.SHEETNAME, model.questionRow, 0);
```

```
                    model.wrongQuestionCount++;
                }

                //retrieve answer text
                String answerAText = model.retrieveAnswerAText();
                String answerBText = model.retrieveAnswerBText();
                String answerCText = model.retrieveAnswerCText();
                String answerDText = model.retrieveAnswerDText();

                //display answer text
                view.setQuestionTextArea(questionText);
                view.setAnswerAText(answerAText);
                view.setAnswerBText(answerBText);
                view.setAnswerCText(answerCText);
                view.setAnswerDText(answerDText);
            }
        }
    }

    private class quizDurationDropBoxListener implements
ActionListener{
        /**
         * listener to listen what the user selects from the drop box
         * checks if the necessary requirements are met for the user to
start the quiz
         * @param e the event to be processed when the user hovers the
drop box
         */
        @Override
        public void actionPerformed(ActionEvent e){
            if(e.getSource() == view.quizDuration){
                String tempHolder = (String)
view.quizDuration.getSelectedItem();
                model.durationInMinutes =
Integer.parseInt(tempHolder);
```

```
                    model.isDurationSelected = true;

                    //enable the start quiz button if both topics AND a
duration are selected
                    if(model.isTopicsSelected == true)
view.startQuizButton.setEnabled(true);
               }
          }
     }

    //METHOD LISTENERS FOR TOPICS
    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
          * @param e the event to be processed when the user checks this
radiobutton
          */
         @Override
         public void actionPerformed(ActionEvent e){
              if(e.getSource() == view.kinematics &&
view.kinematics.isSelected()){ //checks if this has already been
selected
                    if(!model.topics.contains("K")) model.topics.add("K");
//only add if the topics is not already in the list
                    if(model.topics.size()>=3){ //if this condition is met
to start the quiz
                         model.isTopicsSelected = true;

                         //enable the start quiz button if both topics AND
a duration are selected
                         if(model.isDurationSelected == true)
view.startQuizButton.setEnabled(true);
                    }
```

```
                }
                else if(e.getSource() == view.kinematics &&
!view.kinematics.isSelected()){ //checks if this has been unselected
                    if(model.topics.contains("K"))
model.topics.remove("K");

                    //disable the start qui button if the requirements
arent met
                    if(model.topics.size()<3){
                        model.isTopicsSelected = false;
                        view.startQuizButton.setEnabled(false);
                    }
                }
            }
    }

    //ALL OF THE FOLLOWING 7 LISTENERS HAVE THE SAME LOGIC AND CODE AS
THE ONE ABOVE
    private class forcesAndMomentumListener implements ActionListener{
        /**
         * manipulates what topics the user has selected
         * checks if the requirements are met for the user to start the
quiz
         * @param e the event to be processed when the user checks this
radiobutton
         */
        @Override
        public void actionPerformed(ActionEvent e){
            if(e.getSource() == view.forcesAndMomentum &&
view.forcesAndMomentum.isSelected()){ //checks if this has already
been selected
                    if(!model.topics.contains("FAM"))
model.topics.add("FAM");
                    if(model.topics.size()>=3){
                        model.isTopicsSelected = true;
                        if(model.isDurationSelected == true)
view.startQuizButton.setEnabled(true);
                   }
              }
              else if(e.getSource() == view.forcesAndMomentum &&
!view.forcesAndMomentum.isSelected()){
                   if(model.topics.contains("FAM"))
model.topics.remove("FAM");
                   if(model.topics.size()<3){
                        model.isTopicsSelected = false;
                        view.startQuizButton.setEnabled(false);
                   }
              }
         }
    }

    private class workPowerEnergyListener implements ActionListener{
         /**
          * manipulates what topics the user has selected
          * checks if the requirements are met for the user to start the
quiz
          * @param e the event to be processed when the user checks this
radiobutton
          */
         @Override
         public void actionPerformed(ActionEvent e){
              if(e.getSource() == view.workPowerEnergy &&
view.workPowerEnergy.isSelected()){ //checks if this has already been
selected
                   if(!model.topics.contains("WPE"))
model.topics.add("WPE");
                   if(model.topics.size()>=3){
                        model.isTopicsSelected = true;
                        if(model.isDurationSelected == true)
view.startQuizButton.setEnabled(true);
```

```
                }
            }
            else if(e.getSource() == view.workPowerEnergy &&
!view.workPowerEnergy.isSelected()){
                if(model.topics.contains("WPE"))
model.topics.remove("WPE");
                if(model.topics.size()<3){
                    model.isTopicsSelected = false;
                    view.startQuizButton.setEnabled(false);
                }
            }
        }
    }

    private class rotationalMechanicsListener implements
ActionListener{
        /**
         * manipulates what topics the user has selected
         * checks if the requirements are met for the user to start the
quiz
         * @param e the event to be processed when the user checks this
radiobutton
         */
        @Override
        public void actionPerformed(ActionEvent e){
            if(e.getSource() == view.rotationalMechanics &&
view.rotationalMechanics.isSelected()){ //checks if this has already
been selected
                if(!model.topics.contains("RM"))
model.topics.add("RM");
                if(model.topics.size()>=3){
                    model.isTopicsSelected = true;
                    if(model.isDurationSelected == true)
view.startQuizButton.setEnabled(true);
                }
            }
            else if(e.getSource() == view.rotationalMechanics &&
!view.rotationalMechanics.isSelected()){
                if(model.topics.contains("RM"))
model.topics.remove("RM");
                if(model.topics.size()<3){
                    model.isTopicsSelected = false;
                    view.startQuizButton.setEnabled(false);
                }
            }
        }
    }

   private class thermalEnergyTransferListener implements
ActionListener{
        /**
         * manipulates what topics the user has selected
         * checks if the requirements are met for the user to start the
quiz
         * @param e the event to be processed when the user checks this
radiobutton
         */
        @Override
        public void actionPerformed(ActionEvent e){
            if(e.getSource() == view.thermalEnergyTransfer &&
view.thermalEnergyTransfer.isSelected()){ //checks if this has
already been selected
                if(!model.topics.contains("TET"))
model.topics.add("TET");
                if(model.topics.size()>=3){
                    model.isTopicsSelected = true;
                    if(model.isDurationSelected == true)
view.startQuizButton.setEnabled(true);
                }
            }
```

```
           else if(e.getSource() == view.thermalEnergyTransfer &&
!view.thermalEnergyTransfer.isSelected()){
               if(model.topics.contains("TET"))
model.topics.remove("TET");
               if(model.topics.size()<3){
                   model.isTopicsSelected = false;
                   view.startQuizButton.setEnabled(false);
               }
           }
       }
   }

   private class greenhouseEffectListener implements ActionListener{
       /**
        * manipulates what topics the user has selected
        * checks if the requirements are met for the user to start the
quiz
        * @param e the event to be processed when the user checks this
radiobutton
        */
       @Override
       public void actionPerformed(ActionEvent e){
           if(e.getSource() == view.greenhouseEffect &&
view.greenhouseEffect.isSelected()){ //checks if this has already
been selected
               if(!model.topics.contains("GE"))
model.topics.add("GE");
               if(model.topics.size()>=3){
                   model.isTopicsSelected = true;
                   if(model.isDurationSelected == true)
view.startQuizButton.setEnabled(true);
               }
           }
           else if(e.getSource() == view.greenhouseEffect &&
!view.greenhouseEffect.isSelected()){
```

```
                    if(model.topics.contains("GE"))
model.topics.remove("GE");
                    if(model.topics.size()<3){
                         model.isTopicsSelected = false;
                         view.startQuizButton.setEnabled(false);
                    }
               }
          }
     }

    private class idealGasModelListener implements ActionListener {
         /**
          * manipulates what topics the user has selected
          * checks if the requirements are met for the user to start the
quiz
          * @param e the event to be processed when the user checks this
radiobutton
          */
         @Override
         public void actionPerformed(ActionEvent e) {
              if (e.getSource() == view.idealGasModel &&
view.idealGasModel.isSelected()) { //checks if this has already been
selected
                   if (!model.topics.contains("IGM"))
model.topics.add("IGM");
                   if(model.topics.size()>=3){
                        model.isTopicsSelected = true;
                        if(model.isDurationSelected == true)
view.startQuizButton.setEnabled(true);
                   }
              }
              else if(e.getSource() == view.idealGasModel &&
!view.idealGasModel.isSelected()){
                   if(model.topics.contains("IGM"))
model.topics.remove("IGM");
```

```
                if(model.topics.size()<3){
                    model.isTopicsSelected = false;
                    view.startQuizButton.setEnabled(false);
                }
            }
        }
    }

    private class thermalDynamicsListener implements ActionListener{
        /**
         * manipulates what topics the user has selected
         * checks if the requirements are met for the user to start the
quiz
         * @param e the event to be processed when the user checks this
radiobutton
         */
        @Override
        public void actionPerformed(ActionEvent e){
            if(e.getSource() == view.thermalDynamics &&
view.thermalDynamics.isSelected()){ //checks if this has already been
selected
                if(!model.topics.contains("TD"))
model.topics.add("TD");
                if(model.topics.size()>=3){
                    model.isTopicsSelected = true;
                    if(model.isDurationSelected == true)
view.startQuizButton.setEnabled(true);
                }
            }
            else if(e.getSource() == view.thermalDynamics &&
!view.thermalDynamics.isSelected()){
                if(model.topics.contains("TD"))
model.topics.remove("TD");
                if(model.topics.size()<3){
                    model.isTopicsSelected = false;
                    view.startQuizButton.setEnabled(false);
                }
            }
        }
    }

    //METHOD LISTENERS FOR ANSWER BUTTONS
    private class answerAButtonListener implements ActionListener{
        /**
         * checks the answer selected by the user with the correct
answer
         * @param e the event to be processed when the user checks this
button
         */
        @Override
        public void actionPerformed(ActionEvent e){
            if(e.getSource() == view.answerA){ //checks the picked
answer with the actual answer
                model.answerChoicePicked = "A";
                boolean isCorrectAnswer = model.checkAnswer();

                if(isCorrectAnswer){
                    view.answerChoiceAPicked();
                    view.nextButton.setEnabled(true);
                }
                else view.answerA.setBackground(Color.red);

                model.alreadyAnswered = true;
                view.answerA.setEnabled(false);
            }
        }
    }

    private class answerBButtonListener implements ActionListener{
        /**
```

```
         * checks the answer selected by the user with the correct
answer
         * @param e the event to be processed when the user checks this
button
         */
        @Override
        public void actionPerformed(ActionEvent e){
            if(e.getSource() == view.answerB){ //checks the picked
answer with the actual answer
                model.answerChoicePicked = "B";
                boolean isCorrectAnswer = model.checkAnswer();

                if(isCorrectAnswer){
                    view.answerChoiceBPicked();
                    view.nextButton.setEnabled(true);
                }
                else view.answerB.setBackground(Color.red);

                model.alreadyAnswered = true;
                view.answerB.setEnabled(false);
            }
        }
    }

    private class answerCButtonListener implements ActionListener{
        /**
         * checks the answer selected by the user with the correct
answer
         * @param e the event to be processed when the user checks this
button
         */
        @Override
        public void actionPerformed(ActionEvent e){ //checks the
picked answer with the actual answer
            if(e.getSource() == view.answerC){
                model.answerChoicePicked = "C";
                boolean isCorrectAnswer = model.checkAnswer();

                if(isCorrectAnswer){
                    view.answerChoiceCPicked();
                    view.nextButton.setEnabled(true);
                }
                else view.answerC.setBackground(Color.red);

                model.alreadyAnswered = true;
                view.answerC.setEnabled(false);
            }
        }
    }

    private class answerDButtonListener implements ActionListener{
        /**
         * checks the answer selected by the user with the correct
answer
         * @param e the event to be processed when the user checks this
button
         */
        @Override
        public void actionPerformed(ActionEvent e){ //checks the
picked answer with the actual answer
            if(e.getSource() == view.answerD){
                model.answerChoicePicked = "D";
                boolean isCorrectAnswer = model.checkAnswer();

                if(isCorrectAnswer){
                    view.answerChoiceDPicked();
                    view.nextButton.setEnabled(true);
                }
                else view.answerD.setBackground(Color.red);
```

```
                    model.alreadyAnswered = true;
                    view.answerD.setEnabled(false);
                }
            }
        }
}
```

Code - ExcelReader Class

```
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class ExcelReader {

    /**
     * retrieves a text from an excel file according to the parameters
     * @param sheetName is the name of sheet in the excel file the
program is going to search
     * @param rNum is the row number in the excel file the program is
going to search starting at 0
     * @param cNum is the cell number in the excel file the program is
going to search starting at 0
     * @return the string of the text in sheet 'sheetName', at row
'rNum', and at cell 'cNum' from the excel file
     */
   public String readExcel(String sheetName, int rNum, int cNum){
         String data = "";
         String excelLocation = "C:\\College\\1 IB\\CS HL\\IA\\IA
product\\Test.xlsx";
         try {
               //create a workbook from the excel file linked
```

```
            FileInputStream fis = new FileInputStream(excelLocation);
            XSSFWorkbook wb = new XSSFWorkbook(fis);

            //get the necessary information to retrieve from the
workbook created previously
            Sheet sheet = wb.getSheet(sheetName);
            Row row = sheet.getRow(rNum);
            Cell cell = row.getCell(cNum);
            data = cell.getStringCellValue();
        } catch (Exception e) {
            System.err.println("Error reading Excel file: " +
excelLocation);
            e.printStackTrace();
        }
        return data;
    }
}
```

Code - CountdownTimer Class

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CountdownTimer {
   //attributes
   private View view;
   private Model model;
   private DrawGraph drawGraph;
   private int minutes = 0;
   private int seconds = 0;
   Timer timer;
   JLabel timerLabel = new JLabel("");



   //constructor
```

```
    public CountdownTimer(Model model, View view, DrawGraph
drawGraph){
        this.timerLabel = timerLabel;
        this.model = model;
        this.view = view;
        this.drawGraph = drawGraph;
        timerLabel.setBounds(900, 53, 500, 30);
    }


    //methods

    /**
     * This starts the timer
     */
    public void startTimer(){
        minutes = model.durationInMinutes; //total number of minutes
        int totalSeconds = minutes*60 + seconds + 1; //total number of
seconds
        timer = new Timer(1000, new ActionListener() {
            int count = totalSeconds;
            @Override
            public void actionPerformed(ActionEvent e) {
                count--;
                if(count >= 0){ //if the timer is still running
                    int remainingMinutes = count/60;
                    int remainingSeconds = count%60;
                    timerLabel.setText(String.format("Time Remaining:
%02d:%02d", remainingMinutes, remainingSeconds)); //formatting
                }
                else{
                    timer.stop();

                    //update result page if time runs out
                    int scoreToAdd =
(int)(((double)model.correctAnswers /
(double)model.totalNumberOfQuestions) * 100);
                    model.scores.add(scoreToAdd);
                    drawGraph.showGraph(model.scores, view);
                    view.setTopicsToFocusOnText("You ran out of
time!");
                    view.displayResultPage();
                }
            }
        });
        timer.start();
    }
}
```

Code - DrawGraph Class

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.RenderingHints;
import java.awt.Stroke;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javax.swing.*;

@SuppressWarnings("serial")
public class DrawGraph extends JPanel {
   //attributes
   private View view;
   private static final int MAX_SCORE = 100;
   private static final int PREF_W = 650;
   private static final int PREF_H = 320;
   private static final int BORDER_GAP = 30;
   private static final Color GRAPH_COLOR = Color.green;
   private static final Color GRAPH_POINT_COLOR = new Color(150, 50,
50, 180);
   private static final Stroke GRAPH_STROKE = new BasicStroke(3f);
   private static final int GRAPH_POINT_WIDTH = 12;
   private static final int Y_HATCH_CNT = 100;
   private List<Integer> scores;


   //constructor
   public DrawGraph(List<Integer> scores) {
       this.scores = scores;
   }


   //methods

   /**
    * draws the necessary components of a line graph
    * @param g the <code>Graphics</code> object to protect is a
library class implemented
    */
   @Override
   protected void paintComponent(Graphics g) {
       super.paintComponent(g);
       Graphics2D g2 = (Graphics2D)g;
       g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
```

```
        double xScale = ((double) getWidth() - 2 * BORDER_GAP) /
(scores.size() - 1);
        double yScale = ((double) getHeight() - 2 * BORDER_GAP) /
(MAX_SCORE - 1);

        List<Point> graphPoints = new ArrayList<Point>();
        for (int i = 0; i < scores.size(); i++) {
            int x1 = (int) (i * xScale + BORDER_GAP);
            int y1 = (int) (getHeight() - (scores.get(i) * yScale +
BORDER_GAP));
            graphPoints.add(new Point(x1, y1));
        }

        // create x and y axes
        g2.drawLine(BORDER_GAP, getHeight() - BORDER_GAP, BORDER_GAP,
BORDER_GAP);
        g2.drawLine(BORDER_GAP, getHeight() - BORDER_GAP, getWidth() -
BORDER_GAP, getHeight() - BORDER_GAP);

        // create hatch marks for y axis.
        for (int i = 0; i < Y_HATCH_CNT; i++) {
            int x0 = BORDER_GAP;
            int x1 = GRAPH_POINT_WIDTH + BORDER_GAP;
            int y0 = getHeight() - (((i + 1) * (getHeight() -
BORDER_GAP * 2)) / Y_HATCH_CNT + BORDER_GAP);
            int y1 = y0;
            g2.drawLine(x0, y0, x1, y1);
        }

        // and for x axis
        for (int i = 0; i < scores.size() - 1; i++) {
            int x0 = (i + 1) * (getWidth() - BORDER_GAP * 2) /
(scores.size() - 1) + BORDER_GAP;
            int x1 = x0;
            int y0 = getHeight() - BORDER_GAP;
            int y1 = y0 - GRAPH_POINT_WIDTH;
            g2.drawLine(x0, y0, x1, y1);
        }

        Stroke oldStroke = g2.getStroke();
        g2.setColor(GRAPH_COLOR);
        g2.setStroke(GRAPH_STROKE);
        for (int i = 0; i < graphPoints.size() - 1; i++) {
            int x1 = graphPoints.get(i).x;
            int y1 = graphPoints.get(i).y;
            int x2 = graphPoints.get(i + 1).x;
            int y2 = graphPoints.get(i + 1).y;
            g2.drawLine(x1, y1, x2, y2);
        }

        g2.setStroke(oldStroke);
        g2.setColor(GRAPH_POINT_COLOR);
        for (int i = 0; i < graphPoints.size(); i++) {
            int x = graphPoints.get(i).x - GRAPH_POINT_WIDTH / 2;
            int y = graphPoints.get(i).y - GRAPH_POINT_WIDTH / 2;;
            int ovalW = GRAPH_POINT_WIDTH;
            int ovalH = GRAPH_POINT_WIDTH;
            g2.fillOval(x, y, ovalW, ovalH);
        }
    }

    /**
     * gives the dimensions of the screen
     * @return the dimension of the screen with the width and height
     */
    @Override
    public Dimension getPreferredSize() {
        return new Dimension(PREF_W, PREF_H);
    }
    /**
     * displays the graph onto the GUI
     * @param scores is where all the user's score from the quizzes are
stored
     * @param view is an instance of the class View in order to add the
graph to the GUI
     */
   public static void showGraph(List<Integer> scores, View view) {
        int maxDataPoints = scores.size(); //nb of data that exists
        int maxScore = 100;
        DrawGraph mainPanel = new DrawGraph(scores);
        mainPanel.setBounds(580, 130, PREF_W, PREF_H);
        view.resultPage.add(mainPanel);
        view.resultPage.setComponentZOrder(mainPanel, 0);
    }
}
```