

## Appendix B – Source Code

```

1  import java.awt.EventQueue;
2
3  import javax.swing.JFrame;
4  import javax.swing.JOptionPane;
5  import javax.swing.JPanel;
6  import javax.swing.border.EmptyBorder;
7
8  import javax.swing.JButton;
9
10 import java.awt.event.ActionListener;
11 import java.util.ArrayList;
12 import java.util.LinkedHashMap;
13 import java.util.List;
14 import java.awt.event.ActionEvent;
15 import javax.swing.JComboBox;
16 import javax.swing.JFileChooser;
17 import javax.swing.UIManager;
18
19 import java.io.BufferedReader;
20 import java.io.File;
21 import java.io.FileOutputStream;
22 import java.io.FileReader;
23 import java.io.IOException;
24 import java.io.InputStreamReader;
25 import java.io.PrintWriter;
26 import java.io.Reader;
27 import java.nio.file.Files;
28 import java.nio.file.Paths;
29
30 import org.apache.pdfbox.pdmodel.PDDocument;
31 import org.apache.pdfbox.text.PDFTextStripper;
32 import org.apache.poi.ss.usermodel.Cell;
33 import org.apache.poi.ss.usermodel.Row;
34 import org.apache.poi.ss.usermodel.Sheet;
35 import org.apache.poi.xssf.usermodel.XSSFWorkbook;
36 import javax.swing.DefaultComboBoxModel;
37 import java.awt.event.*;
38 import javax.swing.JLabel;
39 import javax.swing.SwingConstants;
40 import java.awt.Font;
41
42 public class Main extends JFrame {
43
44     public static LinkedHashMap<String, Integer> subjects;
45     public static LinkedHashMap<String, String> abbreviations;
46     public static Main window;
47
48     private static ArrayList<ArrayList<Test>> allTests;
49     private JPanel contentPane;
50     private double timer = 0;
51     private int testsTaken = 0;
52     private double totalScore = 0;
53     private JLabel lblTotalTime;
54     private JLabel lblAverageScore;
55     private JLabel lblAverageTimePerTest;
56     private JLabel lblTotalTests;
57
58
59     public static void main(String[] args) {
60         EventQueue.invokeLater(new Runnable() {
61             public void run() {
62                 try {
63                     Main frame = new Main();
64                     frame.setVisible(true);
65
66                 } catch (Exception e) {
67                     e.printStackTrace();
68                 }
69             }
70         });
71     }
72
73 }
74
75 /**
76  * Create the frame.
77  */
78 @SuppressWarnings("rawtypes")
79 public Main() {
80     WindowListener listener = new WindowAdapter() {
81
82         @Override
83         public void windowClosing(WindowEvent we) {
84             try {
85                 writeToFile();
86                 setVisible(false);
87                 System.exit(0);
88             } catch (IOException e) {
89                 e.printStackTrace();
90             }
91         }
92     };
93     addWindowListener(listener);
94
95     window = this;
96     //Sets the UI Style to the system style
97     try {
98         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
99     } catch (Exception e) {
100         e.printStackTrace();
101     }
102
103     // initializes the allTests arraylist
104     allTests = new ArrayList<ArrayList<Test>>();
105     for (int i = 0; i < 7; i++) {
106         allTests.add(new ArrayList<Test>());
107     }

```

```

107
108
109 //initializes and populates the dictionaries
110 subjects = new LinkedHashMap<String, Integer>();
111 createSubjectDictionary();
112 abbreviations = new LinkedHashMap<String, String>();
113 createAbbreviationsDictionary();
114
115 setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
116 setBounds(100, 100, 852, 480);
117 contentPane = new JPanel();
118 contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
119
120 setContentPane(contentPane);
121 contentPane.setLayout(null);
122
123 JComboBox testSelector = new JComboBox();
124 testSelector.setEnabled(false);
125 testSelector.setBounds(10, 44, 424, 22);
126 contentPane.add(testSelector);
127
128 JComboBox subjectSelector = new JComboBox(getSubjects());
129 subjectSelector.addActionListener(new ActionListener() {
130     public void actionPerformed(ActionEvent e) {
131         String a = (String) subjectSelector.getSelectedItem();
132
133         if (!a.equals("Choose Subject")) {
134             int num = subjects.get(abbreviations.get(a));
135             testSelector.setModel(setTestSelectorOptions(num));
136
137             testSelector.setEnabled(true);
138         } else {
139             testSelector.setEnabled(false);
140             testSelector.setModel(new DefaultComboBoxModel());
141         }
142     }
143 });
144
145 subjectSelector.setBounds(10, 11, 424, 22);
146 contentPane.add(subjectSelector);
147
148 JButton btnQuiz = new JButton("Quiz");
149 btnQuiz.addActionListener(new ActionListener() {
150     public void actionPerformed(ActionEvent e) {
151         int sub = subjectSelector.getSelectedIndex() - 1;
152         int test = testSelector.getSelectedIndex();
153         if (sub == -1 || test == -1)
154             JOptionPane.showMessageDialog(null, "No Test Selected", "Error!", JOptionPane.ERROR_MESSAGE);
155         else {
156             QuizMenu quiz = new QuizMenu(allTests.get(sub).get(test));
157             quiz.setVisible(true);
158             setVisible(false);
159         }
160     }
161 });
162 btnQuiz.setBounds(10, 131, 189, 29);
163 contentPane.add(btnQuiz);
164
165 JButton btnReview = new JButton("Review Missed Questions");
166 btnReview.addActionListener(new ActionListener() {
167     public void actionPerformed(ActionEvent e) {
168         int sub = subjectSelector.getSelectedIndex() - 1;
169         int test = testSelector.getSelectedIndex();
170         if (sub == -1 || test == -1)
171             JOptionPane.showMessageDialog(null, "No Test Selected", "Error!", JOptionPane.ERROR_MESSAGE);
172         else {
173             Test review = allTests.get(sub).get(test).getReviewTest();
174             if (review.getQuestions().size() == 0) {
175                 JOptionPane.showMessageDialog(null, "Test has nothing to review :D", "Error!",
176                     JOptionPane.ERROR_MESSAGE);
177             } else {
178                 QuizMenu quiz = new QuizMenu(allTests.get(sub).get(test).getReviewTest());
179                 quiz.setVisible(true);
180                 setVisible(false);
181             }
182         }
183     }
184 });
185
186 btnReview.setBounds(10, 171, 189, 29);
187 contentPane.add(btnReview);
188
189 JButton btnExit = new JButton("Exit");
190 btnExit.addActionListener(new ActionListener() {
191     public void actionPerformed(ActionEvent e) {
192         try {
193             writeToFile();
194             setVisible(false);
195             System.exit(0);
196         } catch (IOException el) {
197         }
198     }
199 });
200 btnExit.setBounds(10, 294, 189, 29);
201
202 contentPane.add(btnExit);
203
204 JButton btnImport = new JButton("Import");
205 btnImport.addActionListener(new ActionListener() {
206     public void actionPerformed(ActionEvent e) {
207         ImportMenu menu = new ImportMenu();
208         menu.setVisible(true);
209     }
210 });
211 btnImport.setBounds(10, 212, 189, 29);
212 contentPane.add(btnImport);
213
214 JButton btnExport = new JButton("Export");
215 btnExport.addActionListener(new ActionListener() {

```

```

214     btnExport.addActionListener(new ActionListener() {
215     public void actionPerformed(ActionEvent e) {
216         int sub = subjectSelector.getSelectedIndex() - 1;
217         int test = testSelector.getSelectedIndex();
218         if (sub == -1)
219             JOptionPane.showMessageDialog(null, "No Test Selected", "Error!", JOptionPane.ERROR_MESSAGE);
220         else
221         {
222             int value = JOptionPane.showOptionDialog(null, "Would you like to export the currently selected test or all tests in this subject", "Export Optio
223                 JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, new String[] { "Only Selected Test", "All Tests"}, null);
224             if (value == -1)
225                 return;
226             else
227             {
228                 String fileLocation = selectDir();
229                 if (fileLocation == null)
230                     return;
231                 if (value == JOptionPane.YES_OPTION)
232                     exportFile(allTests.get(sub).get(test), 20, fileLocation);
233                 else
234                 {
235                     for (int i = 0; i < testSelector.getItemCount(); i++) {
236                         exportFile(allTests.get(sub).get(i), 20, fileLocation);
237                         System.out.println("Done with test: " + i);
238                     }
239                 }
240             }
241         }
242     }
243 });
244
245 btnExport.setBounds(10, 253, 189, 29);
246 contentPane.add(btnExport);
247
248 JLabel lblStats = new JLabel("Statistics");
249 lblStats.setFont(new Font("Calibri", Font.BOLD, 30));
250 lblStats.setVerticalAlignment(SwingConstants.TOP);
251 lblStats.setHorizontalAlignment(SwingConstants.CENTER);
252 lblStats.setBounds(452, 44, 372, 44);
253 contentPane.add(lblStats);
254
255 lblTotalTime = new JLabel("Total Time: ");
256 lblTotalTime.setFont(new Font("Calibri", Font.BOLD, 12));
257 lblTotalTime.setBounds(462, 137, 362, 22);
258 contentPane.add(lblTotalTime);
259
260 lblAverageScore = new JLabel("Average Score:");
261 lblAverageScore.setFont(new Font("Calibri", Font.BOLD, 12));
262 lblAverageScore.setBounds(462, 171, 362, 22);
263 contentPane.add(lblAverageScore);
264
265 lblAverageTimePerTest = new JLabel("Average Time per Test:");
266 lblAverageTimePerTest.setFont(new Font("Calibri", Font.BOLD, 12));
267 lblAverageTimePerTest.setBounds(462, 205, 362, 22);
268 contentPane.add(lblAverageTimePerTest);
269
270 lblTotalTests = new JLabel("Tests Taken:");
271 lblTotalTests.setFont(new Font("Calibri", Font.BOLD, 12));
272 lblTotalTests.setBounds(462, 240, 362, 22);
273 contentPane.add(lblTotalTests);
274
275 JButton btnResetStats = new JButton("Reset Statistics");
276 btnResetStats.addActionListener(new ActionListener() {
277     public void actionPerformed(ActionEvent e) {
278         int result = JOptionPane.showConfirmDialog(null, "Are you sure you want to reset all statistics?", "", 0);
279         if (result == JOptionPane.OK_OPTION) {
280             timer = 0;
281             testsTaken = 0;
282             totalScore = 0;
283             updateLabels();
284         }
285     }
286 });
287 btnResetStats.setBounds(499, 403, 294, 26);
288 contentPane.add(btnResetStats);
289
290 // see if there was previously saved data in current directory
291 if (new File("data.txt").exists()) {
292     //attempts to import the data
293     try {
294         importExistingTests();
295         System.out.println("Tests Imported");
296         // If errors, it just continue on
297     } catch (IOException e) {
298         e.printStackTrace();
299         JOptionPane.showMessageDialog(null,
300             "A problem occurred with loading the data. Continuing without importing data.", "Error!",
301             JOptionPane.ERROR_MESSAGE);
302     }
303 }
304
305 updateLabels();
306
307 /**
308  * Gets the subjects.
309  *
310  * @return all subjects
311  */
312 private String[] getSubjects() {
313     String[] selections = new String[8];
314     selections[0] = "Choose Subject";
315
316     String[] objs = abbreviations.keySet().toArray(new String[0]);
317
318

```

```

322     for (int i = 0; i < objs.length; i++) {
323         selections[i + 1] = objs[i];
324     }
325
326     return selections;
327 }
328
329 /**
330  * Import tests from Pdf files
331  *
332  * @param questions the location of the pdf that contains the questions
333  * @param answers   the location of the pdf that contains the answers
334  * @param subject   the subject the tests should be added to
335  * @throws IOException Signals that an I/O exception has occurred.
336  */
337 public static void importTestsPdf(File questions, File answers, String subject) throws IOException {
338     ArrayList<Test> subjectTests = new ArrayList<Test>();
339     PDDocument document = PDDocument.load(questions);
340     PDFTextStripper pdfStripper = new PDFTextStripper();
341     String text = pdfStripper.getText(document);
342     document.close();
343
344     String[] testsArr;
345     // Splits the text into individual tests
346     if (text.contains("FOCUSED QUIZ"))
347         testsArr = text.split("\\v(" + subject + ")\\s(\\v|F)");
348     else
349         testsArr = text.split("\\v(" + subject + ")\\s\\v");
350
351     // puts into a matrix of tests and lines in tests
352     String[][] parts = new String[testsArr.length - 1][];
353     for (int i = 1; i < testsArr.length; i++)
354         parts[i - 1] = testsArr[i].split("\\n");
355
356     // gets data for the questions
357     for (int i = 0; i < parts.length; i++) {
358         String[] currTestArr = parts[i];
359         int j = 0;
360         if (currTestArr[j].equals(" ") || currTestArr[j].equals(""))
361             j++;
362         String id = currTestArr[j].substring(currTestArr[j].length() - 4, currTestArr[j].length() - 2);
363         String testTitle = "";
364         j++;
365
366         while (currTestArr[j].charAt(0) != ' ')
367             testTitle += currTestArr[j++].strip() + " ";
368         Test currTest = new Test(subject, id, testTitle);
369         j++;
370
371         while (j < currTestArr.length) {
372             String question = "";
373             String a = "";
374             String b = "";
375             String c = "";
376             String d = "";
377             String e = "";
378             // Get Question
379             while (!currTestArr[j].substring(0, 2).equals("a."))
380                 question += currTestArr[j++].strip() + " ";
381
382             // Get Choice A
383             while (!currTestArr[j].substring(0, 2).equals("b."))
384                 a += currTestArr[j++].strip() + " ";
385             // Get Choice B
386             while (!currTestArr[j].substring(0, 2).equals("c."))
387                 b += currTestArr[j++].strip() + " ";
388             // Get Choice C
389             while (!currTestArr[j].substring(0, 2).equals("d."))
390                 c += currTestArr[j++].strip() + " ";
391             // Get Choice D
392             while (!currTestArr[j].substring(0, 2).equals("e."))
393                 d += currTestArr[j++].strip() + " ";
394             // Get Choice E
395             // REGEX: digit 1-9 + period OR 2 digits
396             while (j < currTestArr.length && !currTestArr[j].substring(0, 2).matches("[1-9]\\.|\\d\\d")) {
397                 if (currTestArr[j].contains("DEMIDEC"))
398                     j += 2;
399                 else
400                     e += currTestArr[j++].strip() + " ";
401             }
402
403             Question q = new Question(question.strip(), a, b, c, d, e);
404             currTest.addQuestion(q);
405         }
406
407         subjectTests.add(currTest);
408     }
409     document = PDDocument.load(answers);
410     text = pdfStripper.getText(document);
411     document.close();
412
413     // Splits the text into tests
414     testsArr = text.split("\\v(" + subject + ")\\s(\\v|F)");
415     // puts into a matrix of tests and lines in tests
416     parts = new String[testsArr.length - 1][];
417
418     for (int i = 1; i < testsArr.length; i++)
419         parts[i - 1] = testsArr[i].split("\\n");
420
421     // gets data from answers
422     for (int i = 0; i < parts.length; i++) {
423         int questionNum = 0;
424         String[] currTestArr = parts[i];
425
426         int j = 0;

```

```

429     if (currTestArr[j].equals(" "))
430         j++;
431
432     while (currTestArr[j].length() != 2)
433         j++;
434     j++;
435
436     while (j < currTestArr.length) {
437         String currAnswer = "";
438
439         do {
440             if (currTestArr[j].matches("\\s\\s"))
441                 j++;
442             else if (currTestArr[j].contains("DEMIDEC"))
443                 j += 2;
444             else
445                 currAnswer += currTestArr[j++].strip() + " ";
446         } while (j < currTestArr.length && !currTestArr[j].matches("\\s\\s") && (currTestArr[j].length() <= 5
447             || !currTestArr[j].substring(0, 5).matches("[1-9]\\s[A-Z]\\s|\\d\\d\\s[A-Z]"));
448
449         if (currAnswer.length() != 0) {
450             int index = currAnswer.indexOf('.');
451             String ans = currAnswer.substring(index + 2, index + 3);
452             int end = currAnswer.indexOf('[');
453             String ansExp = currAnswer.substring(index + 4);
454             if (end != -1)
455                 ansExp = currAnswer.substring(index + 4, end);
456             subjectTests.get(i).getSpecificQuestion(questionNum).setAnswer(ans, ansExp);
457             questionNum++;
458         }
459     }
460 }
461 }
462 System.out.println("Imported Tests");
463 int num = subjects.get(subject);
464 allTests.set(num, subjectTests);
465 }
466
467 /**
468  * Import existing tests from data.txt.
469  *
470  * @throws IOException Signals that an I/O exception has occurred.
471  */
472 private void importExistingTests() throws IOException {
473     String[] keys = subjects.keySet().toArray(new String[0]);
474     String[] subs = new String[keys.length + 1];
475     System.arraycopy(keys, 0, subs, 0, keys.length);
476     subs[subs.length - 1] = "END";
477     List<String> content = Files.readAllLines(Paths.get("data.txt"));
478
479     int j = 0;
480     for (int i = 0; i < subs.length - 1; i++) {
481         ArrayList<Test> subjectTests = new ArrayList<Test>();
482         int testNum = 0;
483         j += 2;
484         while (j < content.size() - 3 && i + 1 < subs.length && !content.get(j).equals(subs[i + 1])) {
485
486             String testTitle = content.get(j++);
487             Test currTest = new Test(subs[i], Integer.toString(testNum), testTitle);
488             while (!content.get(j).equals("")) {
489
490                 String question = content.get(j++);
491                 String a = content.get(j++);
492                 String b = content.get(j++);
493                 String c = content.get(j++);
494                 String d = content.get(j++);
495                 String e = content.get(j++);
496                 String ans = content.get(j++);
497                 String exp = content.get(j++);
498                 String score = content.get(j++);
499
500                 currTest.addQuestion(new Question(question, a, b, c, d, e, ans, exp, score));
501             }
502             subjectTests.add(currTest);
503             testNum++;
504             j++;
505         }
506         allTests.set(i, subjectTests);
507     }
508     j++;
509     timer = Integer.parseInt(content.get(j++));
510     testsTaken = Integer.parseInt(content.get(j++));
511     totalScore = Integer.parseInt(content.get(j++));
512
513 }
514
515 /**
516  * Write to file.
517  *
518  * @throws IOException Signals that an I/O exception has occurred.
519  */
520 private void writeToFile() throws IOException {
521     PrintWriter pw = new PrintWriter("data.txt", "UTF-8");
522     String[] arr = subjects.keySet().toArray(new String[0]);
523     for (int i = 0; i < arr.length; i++) {
524         pw.write(arr[i] + "\\n");
525
526         ArrayList<Test> currSubject = allTests.get(i);
527
528         for (Test test : currSubject) {
529             ArrayList<Question> quests = test.getQuestions();
530             pw.write("\\n" + test.getTestName() + "\\n");
531
532             for (Question q : quests) {
533                 String[] str = q.getQuestion();
534                 for (String str : str) {

```

```

536         pw.write(str + "\n");
537     }
538     pw.write(q.getScore() + "\n");
539 }
540 }
541 pw.write('\n');
542 }
543 pw.write("END\n");
544 pw.write(Integer.toString((int) timer) + '\n');
545 pw.write(Integer.toString(testsTaken) + '\n');
546 pw.write(Integer.toString((int) totalScore) + '\n');
547 System.out.println("Wrote to File!");
548 pw.close();
549 }
550 }
551
552 /**
553  * Sets the test selector options.
554  *
555  * @param sub the subject that is selected
556  * @return the DefaultComboBoxModel listing all the tests for JComboBox
557  */
558 private DefaultComboBoxModel<String> setTestSelectorOptions(int sub) {
559     ArrayList<Test> curr = allTests.get(sub);
560     DefaultComboBoxModel<String> model = new DefaultComboBoxModel<String>();
561
562     for (Test test : curr) {
563         model.addElement(test.toString());
564     }
565     return model;
566 }
567
568 /**
569  * Exports the file to an xlsx format
570  *
571  * @param test Test that is wanted to be exported
572  * @param time The time that should be allocated to every question
573  */
574 private void exportFile(Test test, int time, String fileLocation) {
575
576     ArrayList<Question> questions = test.getQuestions();
577     XSSFWorkbook workbook = new XSSFWorkbook();
578     Sheet sheet = workbook.createSheet();
579
580     int rows = 0;
581     Row row = sheet.createRow(rows++);
582     String[] header = new String[] { "Question", "Answer 1", "Answer 2", "Answer 3", "Answer 4", "Time",
583         "Correct" };
584     for (int i = 0; i < 7; i++) {
585         Cell cell = row.createCell(i);
586         cell.setCellValue(header[i]);
587     }
588
589     for (Question quest : questions) {
590         if (quest.canKahoot()) {
591             row = sheet.createRow(rows++);
592             String[] q = quest.getFilteredQuestion();
593             for (int i = 0; i < 7; i++) {
594                 Cell cell = row.createCell(i);
595                 if (i < 5)
596                     cell.setCellValue(q[i]);
597                 else if (i == 6) {
598                     cell.setCellValue(q[i - 1]);
599                 } else
600                     cell.setCellValue(time);
601             }
602         }
603     }
604
605     try {
606         // CLEAN FILES WITH \/:?*<>|
607         fileLocation += test.toString().replaceAll("\\\\|\\/|\\:|\\*|\\?|\\<|\\>|\\||", "");
608         FileOutputStream outputStream = new FileOutputStream(fileLocation + ".xlsx");
609         workbook.write(outputStream);
610         workbook.close();
611         outputStream.close();
612         System.out.println("File has been saved");
613     } catch (Exception e) {
614         e.printStackTrace();
615     }
616 }
617
618 /**
619  * Selects the directory the file is saved to
620  *
621  * @return the string containing the directory
622  */
623 private String selectDir() {
624
625     JFileChooser fc = new JFileChooser();
626     fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
627     int output = fc.showOpenDialog(this);
628     if (output == JFileChooser.APPROVE_OPTION) {
629         return fc.getSelectedFile().toString() + "\\";
630     }
631     else
632         JOptionPane.showMessageDialog(null, "Export Cancelled");
633     return null;
634 }
635
636 private void createAbbreviationsDictionary() {
637     abbreviations.put("Art", "ART");
638     abbreviations.put("Economics", "ECON");
639     abbreviations.put("Literature", "LITERATURE");
640     abbreviations.put("Novel", "FICTION");

```

```

643 abbreviations.put("Novel", "LANGLIT");
644 abbreviations.put("Music", "MUSIC");
645 abbreviations.put("Science", "SCIENCE");
646 abbreviations.put("Social Science", "SOCSCI");
647 }
648
649 private void createSubjectDictionary() {
650     subjects.put("ART", 0);
651     subjects.put("ECON", 1);
652     subjects.put("LITERATURE", 2);
653     subjects.put("LANGLIT", 3);
654     subjects.put("MUSIC", 4);
655     subjects.put("SCIENCE", 5);
656     subjects.put("SOCSCI", 6);
657 }
658
659
660 /**
661  * Gets the abbreviations dictionary.
662  *
663  * @return the abbreviations dictionary
664  */
665 public LinkedHashMap<String, String> getAbbreviations() {
666     return abbreviations;
667 }
668
669 /**
670  * Update the statistic private variables
671  *
672  * @param time increments the timer by this time (Given in MM.SS)
673  * @param percent increments the percent by this parameter
674  */
675 public void updateStats(double time, double percent) {
676
677     timer += time;
678     testsTaken++;
679     totalScore += percent;
680     updateLabels();
681 }
682
683
684 /**
685  * Update labels.
686  */
687 private void updateLabels() {
688     lblTotalTime.setText("Total Time: " + (int) timer / 60 + " minutes " + (int) timer % 60 + " seconds");
689     if (testsTaken == 0) {
690         lblAverageTimePerTest.setText("Average Time per Test: No Tests Taken");
691         lblAverageScore.setText("Average Score: No Tests Taken");
692     } else {
693         System.out.println(totalScore);
694         lblAverageScore.setText("Average Score: " + ((int) (totalScore * 100 / testsTaken)) + "%");
695         double avgTime = timer / testsTaken;
696         lblAverageTimePerTest.setText(
697             "Average Time per Test: " + (int) avgTime / 60 + " minutes " + (int) avgTime % 60 + " seconds");
698     }
699     lblTotalTests.setText("Tests Taken: " + testsTaken);
700 }
701 }
702 }

```



```

1  import java.awt.Color;
2  import javax.swing.JFrame;
3  import javax.swing.JPanel;
4  import javax.swing.border.EmptyBorder;
5  import javax.swing.JButton;
6  import java.awt.event.ActionListener;
7  import java.awt.event.WindowAdapter;
8  import java.awt.event.WindowEvent;
9  import java.awt.event.WindowListener;
10 import java.awt.event.ActionEvent;
11 import javax.swing.JLabel;
12 import javax.swing.JOptionPane;
13 import javax.swing.SwingConstants;
14
15 @SuppressWarnings("serial")
16 public class QuizMenu extends JFrame {
17
18     private JPanel contentPane;
19     private Test currTest;
20     private int questionNum = 0;
21     private Question quest;
22     private JLabel[] labels;
23     private JLabel lblExp;
24     private JButton[] buttons;
25     private JButton btnNext;
26     private int score;
27     private long time = System.nanoTime();
28
29     public QuizMenu(Test test) {
30
31         currTest = test;
32         quest = currTest.getSpecificQuestion(questionNum);
33         setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
34         setBounds(100, 100, 852, 480);
35         contentPane = new JPanel();
36         contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
37
38         setContentPane(contentPane);
39         contentPane.setLayout(null);
40
41         WindowListener listener = new WindowAdapter() {
42
43             @Override
44             public void windowClosing(WindowEvent we) {
45
46                 int result = JOptionPane.showConfirmDialog(null, "Are you sure you want to exit this test?", "", 0);
47                 if (result == JOptionPane.OK_OPTION) {
48                     setVisible(false);
49                     Main.window.setVisible(true);
50                     dispose();
51                 }
52             }
53         };
54         addWindowListener(listener);
55
56         lblExp = new JLabel("");
57         lblExp.setVerticalAlignment(SwingConstants.TOP);
58         lblExp.setHorizontalAlignment(SwingConstants.LEFT);
59         lblExp.setBounds(35, 357, 570, 73);
60         contentPane.add(lblExp);
61
62         JLabel lblQuestion = new JLabel("Question");
63         lblQuestion.setHorizontalAlignment(SwingConstants.LEFT);
64         lblQuestion.setBounds(37, 11, 688, 53);
65         contentPane.add(lblQuestion);
66
67         btnNext = new JButton("Next");
68         btnNext.setVisible(false);
69         btnNext.addActionListener(new ActionListener() {
70             public void actionPerformed(ActionEvent e) {
71
72                 questionNum++;
73                 if (currTest.hasNext(questionNum)) {
74                     quest = currTest.getSpecificQuestion(questionNum);
75                     promptQuestion();
76                     btnNext.setVisible(false);

```

```

77     } else {
78         double currTime = (System.nanoTime() - time) / Math.pow(10, 9);
79         JOptionPane.showMessageDialog(null,
80             "Test Completed!\n" + score + " correct out of " + currTest.length() + "\nTime Taken: "
81             + (int) currTime / 60 + " minutes " + (int) currTime % 60 + " seconds",
82             "WOOOOOOO", JOptionPane.PLAIN_MESSAGE);
83
84         Main.window.updateStats(currTime, (double) score / currTest.length());
85         setVisible(false);
86         Main.window.setVisible(true);
87         dispose();
88     }
89 }
90 });
91 btnNext.setBounds(699, 398, 106, 32);
92 contentPane.add(btnNext);
93
94 JLabel lblA = new JLabel("A");
95 lblA.setBounds(135, 75, 351, 45);
96 contentPane.add(lblA);
97
98 JLabel lblB = new JLabel("B");
99 lblB.setBounds(135, 130, 351, 45);
100 contentPane.add(lblB);
101
102 JLabel lblC = new JLabel("C");
103 lblC.setBounds(135, 185, 368, 45);
104 contentPane.add(lblC);
105
106 JLabel lblD = new JLabel("D");
107 lblD.setBounds(135, 240, 368, 45);
108 contentPane.add(lblD);
109
110 JLabel lblE = new JLabel("E");
111 lblE.setBounds(135, 295, 368, 45);
112 contentPane.add(lblE);
113
114 labels = new JLabel[] { lblQuestion, lblA, lblB, lblC, lblD, lblE, lblExp };
115
116
117 JButton btnA = new JButton("A");
118 btnA.addActionListener(new ActionListener() {
119     public void actionPerformed(ActionEvent e) {
120         processGuess('A');
121     }
122 });
123 btnA.setBounds(35, 75, 90, 45);
124 contentPane.add(btnA);
125
126 JButton btnB = new JButton("B");
127 btnB.addActionListener(new ActionListener() {
128     public void actionPerformed(ActionEvent e) {
129         processGuess('B');
130     }
131 });
132 btnB.setBounds(35, 130, 90, 45);
133 contentPane.add(btnB);
134
135 JButton btnC = new JButton("C");
136 btnC.setForeground(new Color(0, 0, 0));
137 btnC.addActionListener(new ActionListener() {
138     public void actionPerformed(ActionEvent e) {
139         processGuess('C');
140     }
141 });
142 btnC.setBounds(35, 185, 89, 45);
143 contentPane.add(btnC);
144
145 JButton btnD = new JButton("D");
146 btnD.addActionListener(new ActionListener() {
147     public void actionPerformed(ActionEvent e) {
148         processGuess('D');
149     }
150 });
151 btnD.setBounds(35, 240, 89, 45);
152 contentPane.add(btnD);
153

```

```

153
154 JButton btnE = new JButton("E");
155 btnE.addActionListener(new ActionListener() {
156     public void actionPerformed(ActionEvent e) {
157         processGuess('E');
158     }
159 });
160 btnE.setBounds(35, 295, 90, 45);
161 contentPane.add(btnE);
162
163 buttons = new JButton[] { btnA, btnB, btnC, btnD, btnE };
164 promptQuestion();
165
166 }
167
168 /**
169  * Disable Answer buttons.
170  */
171 private void disableButtons() {
172     for (JButton button : buttons)
173         button.setEnabled(false);
174 }
175
176 /**
177  * Enable Answer buttons.
178  */
179 private void enableButtons() {
180     for (JButton button : buttons)
181         button.setEnabled(true);
182 }
183
184 /**
185  * Prompt the next question and sets all the labels to correct values.
186  */
187 private void promptQuestion() {
188     reset();
189     String[] data = quest.getQuestion();
190
191     for (int i = 0; i < data.length - 2; i++) {
192         labels[i].setText("<html>" + data[i] + "</html>");
193     }
194 }
195
196 }
197
198 /**
199  * Sets the correct button the green, sets wrong to red.
200  *
201  * @param btn the button to set to green
202  */
203 private void setColors(JButton btn) {
204
205     for (JButton l : buttons) {
206         if (l != btn)
207             l.setBackground(Color.RED);
208         else
209             l.setBackground(Color.GREEN);
210     }
211 }
212
213 /**
214  * Processes the guess.
215  *
216  * @param letter the letter corresponding the the button the user has pressed
217  */
218 private void processGuess(char letter) {
219     if (quest.correct("'" + letter)) {
220         score++;
221         quest.changeScore(true);
222         setColors(buttons[letter - 'A']);
223     } else {
224         setColors(buttons[quest.getQuestion()[6].charAt(0) - 'A']);
225         quest.changeScore(false);
226     }
227     disableButtons();
228     lblExp.setText("<html>" + quest.getQuestion()[7] + "</html>");
229     btnNext.setVisible(true);

```

```
230     }
231
232     /**
233      * Resets the button colors to grey.
234      */
235     private void reset() {
236         for (JButton b : buttons)
237             b.setBackground(Color.GRAY);
238         labels[6].setText("");
239
240         enableButtons();
241     }
242 }
```

```

1 import java.awt.EventQueue;
2
3 import javax.swing.JFrame;
4 import javax.swing.JPanel;
5 import javax.swing.border.EmptyBorder;
6 import javax.swing.filechooser.FileNameExtensionFilter;
7 import javax.swing.JComboBox;
8 import javax.swing.JFileChooser;
9 import javax.swing.JButton;
10 import java.awt.event.ActionListener;
11 import java.io.File;
12 import java.io.IOException;
13 import java.util.LinkedHashMap;
14 import java.awt.event.ActionEvent;
15 import javax.swing.DefaultComboBoxModel;
16 import javax.swing.JLabel;
17 import javax.swing.JOptionPane;
18
19 import java.awt.Color;
20
21 public class ImportMenu extends JFrame{
22
23     private JPanel contentPane;
24     private File questions;
25     private File answers;
26
27     @SuppressWarnings("unchecked")
28     public ImportMenu() {
29         setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
30         setBounds(100, 100, 450, 300);
31         contentPane = new JPanel();
32         contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
33
34         setContentPane(contentPane);
35         contentPane.setLayout(null);
36
37         JLabel lblAnswerFile = new JLabel("No File Chosen");
38         lblAnswerFile.setForeground(Color.RED);
39         lblAnswerFile.setBounds(101, 168, 224, 14);
40         contentPane.add(lblAnswerFile);
41
42         JLabel lblQuestionFile = new JLabel("No File Chosen");
43         lblQuestionFile.setForeground(Color.RED);
44         lblQuestionFile.setBounds(101, 111, 224, 14);
45         contentPane.add(lblQuestionFile);
46
47         JComboBox comboBox = new JComboBox();
48         comboBox.setModel(new DefaultComboBoxModel(
49             new String[] { "Art", "Economics", "Literature", "Novel", "Music", "Science", "Social Science" }));
50         comboBox.setBounds(271, 38, 153, 27);
51         contentPane.add(comboBox);
52
53         JButton btnQuestions = new JButton("Choose File");
54         btnQuestions.addActionListener(new ActionListener() {
55             public void actionPerformed(ActionEvent e) {
56                 File curr = selectFile();
57                 if (curr != null) {
58                     questions = curr;
59                     lblQuestionFile.setText(questions.toString().substring(questions.toString().lastIndexOf("\\")+1));
60                     lblQuestionFile.setForeground(Color.BLACK);
61                 }
62             }
63         });
64         btnQuestions.setBounds(325, 105, 99, 27);
65         contentPane.add(btnQuestions);
66
67         JButton btnAnswers = new JButton("Choose File");
68         btnAnswers.addActionListener(new ActionListener() {
69             public void actionPerformed(ActionEvent e) {
70                 File curr = selectFile();
71                 if (curr != null) {
72                     answers = curr;
73                     lblAnswerFile.setText(answers.toString().substring(answers.toString().lastIndexOf("\\")+1));
74                     lblAnswerFile.setForeground(Color.BLACK);
75                 }
76             }
77         });
78         btnAnswers.setBounds(325, 162, 99, 27);
79         contentPane.add(btnAnswers);
80
81         JLabel lblQuestions = new JLabel("Questions PDF:");
82         lblQuestions.setBounds(10, 105, 89, 27);
83         contentPane.add(lblQuestions);
84
85         JLabel lblAnswersPdf = new JLabel("Answers PDF:");
86         lblAnswersPdf.setBounds(10, 162, 81, 27);
87         contentPane.add(lblAnswersPdf);
88
89         JLabel lblSubjects = new JLabel("Subjects");
90         lblSubjects.setBounds(10, 38, 196, 27);
91         contentPane.add(lblSubjects);
92
93         JButton btnImport = new JButton("Import");
94         btnImport.addActionListener(new ActionListener() {
95             public void actionPerformed(ActionEvent e) {
96                 if (answers == null || questions == null)
97                     JOptionPane.showMessageDialog(null, "No File Selected!", "Error", JOptionPane.ERROR_MESSAGE);
98                 else if (!answers.toString().contains((String) comboBox.getSelectedItem()) || !questions.toString().contains((String) comboBox.getSelectedItem()))
99                     JOptionPane.showMessageDialog(null, "Please check the selected files to ensure you are importing the correct subject!", "Error", JOptionPane.ERROR_MESSAGE);
100                 else
101                 {
102                     try {
103                         Main.importTestsPdf(questions, answers, Main.abbreviations.get(comboBox.getSelectedItem()));
104                     } catch (IOException e1) {
105                         JOptionPane.showMessageDialog(null, "There were problems with reading the file", "Error", JOptionPane.ERROR_MESSAGE);
106                     }
107                 }
108             }
109         });
110     }
111 }

```

```
107,      ,
108      setVisible(false);
109      dispose();
110  }
111  }
112  });
113  btnImport.setBounds(104, 227, 89, 23);
114  contentPane.add(btnImport);
115
116  JButton btnCancel = new JButton("Cancel");
117  btnCancel.addActionListener(new ActionListener() {
118      public void actionPerformed(ActionEvent e) {
119          setVisible(false);
120          dispose();
121      }
122  });
123  btnCancel.setBounds(226, 227, 89, 23);
124  contentPane.add(btnCancel);
125
126  }
127
128  /**
129   * Select file through JFileChooser
130   *
131   * @return the file
132   */
133  private File selectFile() {
134      JFileChooser jc = new JFileChooser();
135      FileNameExtensionFilter filter = new FileNameExtensionFilter(
136          "Pdfs", "pdf");
137      jc.setFileFilter(filter);
138      jc.showOpenDialog(this);
139      File file = jc.getSelectedFile();
140      return file;
141  }
142  }
143
144
145
146  }
```

```

1  /**
2   * The Question Class.
3   */
4  public class Question {
5
6      private String myQuestion;
7      private String myA;
8      private String myB;
9      private String myC;
10     private String myD;
11     private String myE;
12
13     /** The answer to the question. */
14     private String myAnswer;
15     /** The explanation why the given answer is correct. */
16     private String myAnswerExplanation;
17     /** The net amount that the user has gotten this question right/wrong. */
18     private int myScore;
19
20
21     /**
22      * Instantiates a new question.
23      *
24      * @param Question the question
25      * @param A Choice A
26      * @param B Choice B
27      * @param C Choice C
28      * @param D Choice D
29      * @param E Choice E
30      * @param ans Answer
31      * @param ansExp Answer Explanation
32      * @param score the score
33      */
34     public Question(String Question, String A, String B, String C, String D, String E, String ans, String ansExp,
35                     String score) {
36         myQuestion = Question;
37         myA = A;
38         myB = B;
39         myC = C;
40         myD = D;
41         myE = E;
42         myAnswer = ans;
43         myAnswerExplanation = ansExp;
44         myScore = Integer.parseInt(score);
45     }
46
47     /**
48      * Instantiates a new question.
49      *
50      * @param Question the question
51      * @param A Choice A
52      * @param B Choice B
53      * @param C Choice C
54      * @param D Choice D
55      * @param E Choice E
56      */
57     public Question(String Question, String A, String B, String C, String D, String E) {
58         myQuestion = Question;
59         myA = A;
60         myB = B;
61         myC = C;
62         myD = D;
63         myE = E;
64         cleanStrings();
65     }
66
67     /**
68      * Gets the question.
69      *
70      * @return the question
71      */
72     public String[] getQuestion() {
73         return new String[] { myQuestion, myA, myB, myC, myD, myE, myAnswer, myAnswerExplanation };
74     }
75
76     /**
77      * Gets the question that only has 4 responses in a random order.
78      * Used to export to other programs such as Kahoot that only accept 4 answer questions.
79      *
80      * @return the filtered question
81      */
82     public String[] getFilteredQuestion() {
83         String[] arr = new String[6];
84         String[] quest = getQuestion();
85
86         arr[0] = quest[0];
87
88         int rand = (int) (Math.random() * 4) + 1;
89
90         arr[rand] = answer();
91         arr[5] = "" + rand;
92         int i = 1;
93         int j = 1;
94         while (i <= 4) {
95             if (i != rand && !quest[j].equals(answer())) {
96                 arr[i] = quest[j];
97                 i++;
98                 j++;
99             } else if (i == rand)
100                 i++;
101             else if (quest[j].equals(answer()))
102                 j++;
103         }
104         return arr;
105     }
106

```

```

107  /**
108   * Gets the Answer.
109   *
110   * @return the answer
111   */
112  private String answer() {
113      if (myAnswer.equals("A"))
114          return myA;
115      else if (myAnswer.equals("B"))
116          return myB;
117      else if (myAnswer.equals("C"))
118          return myC;
119      else if (myAnswer.equals("D"))
120          return myD;
121      else
122          return myE;
123  }
124
125  /**
126   * Checks if the given String matches Correct answer.
127   *
128   * @param guessss The guess
129   * @return true, if correct
130   */
131  public boolean correct(String guess) {
132      if (guess.equals(myAnswer)) {
133          return true;
134      }
135      return false;
136  }
137
138  /**
139   * Sets the answer.
140   *
141   * @param ans the ans
142   * @param ansExp the ans exp
143   */
144  public void setAnswer(String ans, String ansExp) {
145      myAnswer = ans;
146      myAnswerExplanation = ansExp;
147  }
148
149  public String toString() {
150      return myQuestion + "\nA: " + myA + "\nB: " + myB + "\nC: " + myC + "\nD: " + myD + "\nE: " + myE + "\nAnswer: "
151          + myAnswer + " " + myAnswerExplanation;
152  }
153
154  /**
155   * Runs the cleanString method on every text field
156   */
157  private void cleanStrings()
158  {
159      myQuestion = cleanString(myQuestion);
160      myA = cleanString(myA);
161      myB = cleanString(myB);
162      myC = cleanString(myC);
163      myD = cleanString(myD);
164      myE = cleanString(myE);
165  }
166
167  /**
168   * Cleans the string.
169   *
170   * @param str String to be cleaned
171   * @return cleaned string
172   */
173  private String cleanString(String str)
174  {
175      int i = str.indexOf(".");
176      str= str.substring(i+2);
177
178      return str;
179  }
180
181
182  /**
183   * Can export to kahoot.
184   *
185   * @return true, if successful
186   */
187  public boolean canKahoot()
188  {
189      return myQuestion.length() <= 120 && myA.length() <= 75 && myB.length() <= 75 && myC.length() <= 75 && myD.length() <= 75 && myE.length() <= 75;
190  }
191
192  /**
193   * Change score.
194   *
195   * @param correct If the user has gotten the question right. True if Right, False if wrong
196   */
197  public void changeScore(boolean correct)
198  {
199      if(correct)
200          myScore++;
201
202      else
203          myScore--;
204  }
205
206  /**
207   * Gets the score.
208   *
209   * @return the score
210   */
211  public int getScore()
212  {
213      return mvScore;

```



213 }  
214  
215  
216 }  
217

```

1 import java.util.ArrayList;
2
3 public class Test {
4
5     /** The questions. */
6     private ArrayList<Question> questions;
7
8     /** The test ID. */
9     private String myTest;
10
11     /** The test name. */
12     private String myTestName;
13
14     /** The subject. */
15     private String mySubject;
16
17     /**
18      * Instantiates a new test.
19      *
20      * @param subject the subject
21      * @param testId the test id
22      * @param testName the test name
23      */
24     public Test(String subject, String testId, String testName) {
25         questions = new ArrayList<Question>();
26         myTest = testId;
27         myTestName = testName.replaceAll("\\\\|\\/|\\|:|\\|*|\\|?|\\|<|\\|>|\\|\\|\\|\"", "");
28         mySubject = subject;
29     }
30
31     /**
32      * Instantiates a new test.
33      *
34      * @param subject the subject
35      * @param testId the test id
36      * @param testName the test name
37      * @param quests the questions
38      */
39     public Test(String subject, String testId, String testName, ArrayList<Question> quests) {
40         questions = new ArrayList<Question>();
41         myTest = testId;
42         myTestName = testName.replaceAll("\\\\|\\/|\\|:|\\|*|\\|?|\\|<|\\|>|\\|\\|\\|\"", "");
43         questions = quests;
44         mySubject = subject;
45     }
46
47     /**
48      * Gets the test name.
49      *
50      * @return the test name
51      */
52     public String getTestName() {
53         return myTestName;
54     }
55
56     /**
57      * Gets the test ID.
58      *
59      * @return the test ID
60      */
61     public String getTestID() {
62         return myTest;
63     }
64
65     /**
66      * Gets the subject.

```

```

67     *
68     * @return the subject
69     */
70     public String getSubject() {
71         return mySubject;
72     }
73
74     /**
75     * Sets the subject.
76     *
77     * @param subject the new subject
78     */
79     public void setSubject(String subject) {
80         mySubject = subject;
81     }
82
83
84
85     /**
86     * Gets the questions.
87     *
88     * @return the questions
89     */
90     public ArrayList<Question> getQuestions() {
91         return questions;
92     }
93
94     /**
95     * Gets the specific question.
96     *
97     * @param index the index
98     * @return the specific question
99     */
100    public Question getSpecificQuestion(int index) {
101        return questions.get(index);
102    }
103
104    /**
105    * Adds the question.
106    *
107    * @param ques the ques
108    */
109    public void addQuestion(Question ques) {
110        questions.add(ques);
111    }
112
113
114    /**
115    * To string.
116    *
117    * @return the string
118    */
119    public String toString()
120    {
121        return myTestName;
122    }
123
124    /**
125    * Checks if there is a next question.
126    *
127    * @param index the index
128    * @return true, if successful
129    */
130    public boolean hasNext(int index)
131    {
132        return index < questions.size();
133    }

```

```
134
135  /**
136   * Length.
137   *
138   * @return the int of the questions ArrayList
139   */
140  public int length()
141  {
142      return questions.size();
143  }
144
145
146  /**
147   * Gets the review test.
148   *
149   * @return a test that only contains questions with a score < 0
150   */
151  public Test getReviewTest()
152  {
153      ArrayList<Question> reviewQuests = new ArrayList<Question>();
154      for(Question q: questions)
155      {
156          if(q.getScore() < 0)
157              reviewQuests.add(q);
158      }
159
160      return new Test(mySubject, myTest, myTestName, reviewQuests);
161  }
162
163 }
```