eclipse-workspace - a2223330168_PA_Unidad1/src/a2223330168_PA_Tareas/tarea012.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Model Explorer

type filter text

- a2223330168_PA_Unidad1
  - JRE System Library [jre]
  - src
    - a2223330168_PA_ejemplos
    - a2223330168_PA_ejercicios
    - a2223330168_PA_Tareas
      - dualmodestopwatch.java
      - ejemplo_radio.java
      - PruebaChecks.java
      - PruebaCombo.java
      - Sintaxis_radio.java
      - tarea010.java
      - tarea012.java
    - estructuras
    - MCE
      - MultipleChoiceExam.java
    - tarea011
    - tarea09
- Servers
- webejemplo

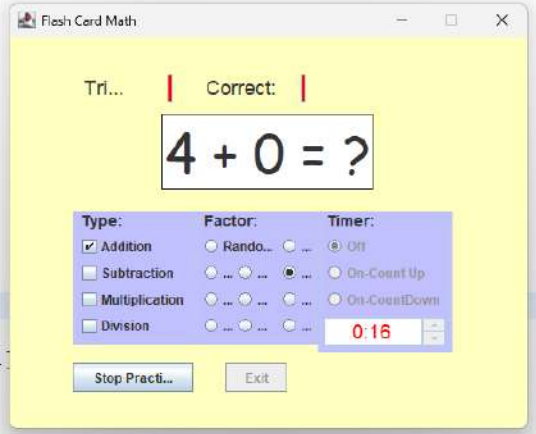
Outline

- tarea012
  - triedLabel : JLabel
  - triedTextField : JTextField
  - correctLabel : JLabel
  - correctTextField : JTextField
  - problemLabel : JLabel
  - dividerLabel : JLabel
  - typePanel : JPanel
  - typeCheckBox : JCheckBox[]
  - factorPanel : JPanel
  - factorButtonGroup : ButtonGroup
  - factorRadioButton : JRadioButton[]
  - timerPanel : JPanel
  - timerButtonGroup : ButtonGroup
  - timerRadioButton : JRadioButton[]
  - timerTextField : JTextField
  - timerScrollBar : JScrollBar
  - startButton : JButton
  - exitButton : JButton
  - problemsTimer : Timer
  - myFont : Font
  - lightBlue : Color
  - myRandom : Random

tarea012.java | MultipleChoiceExam.java

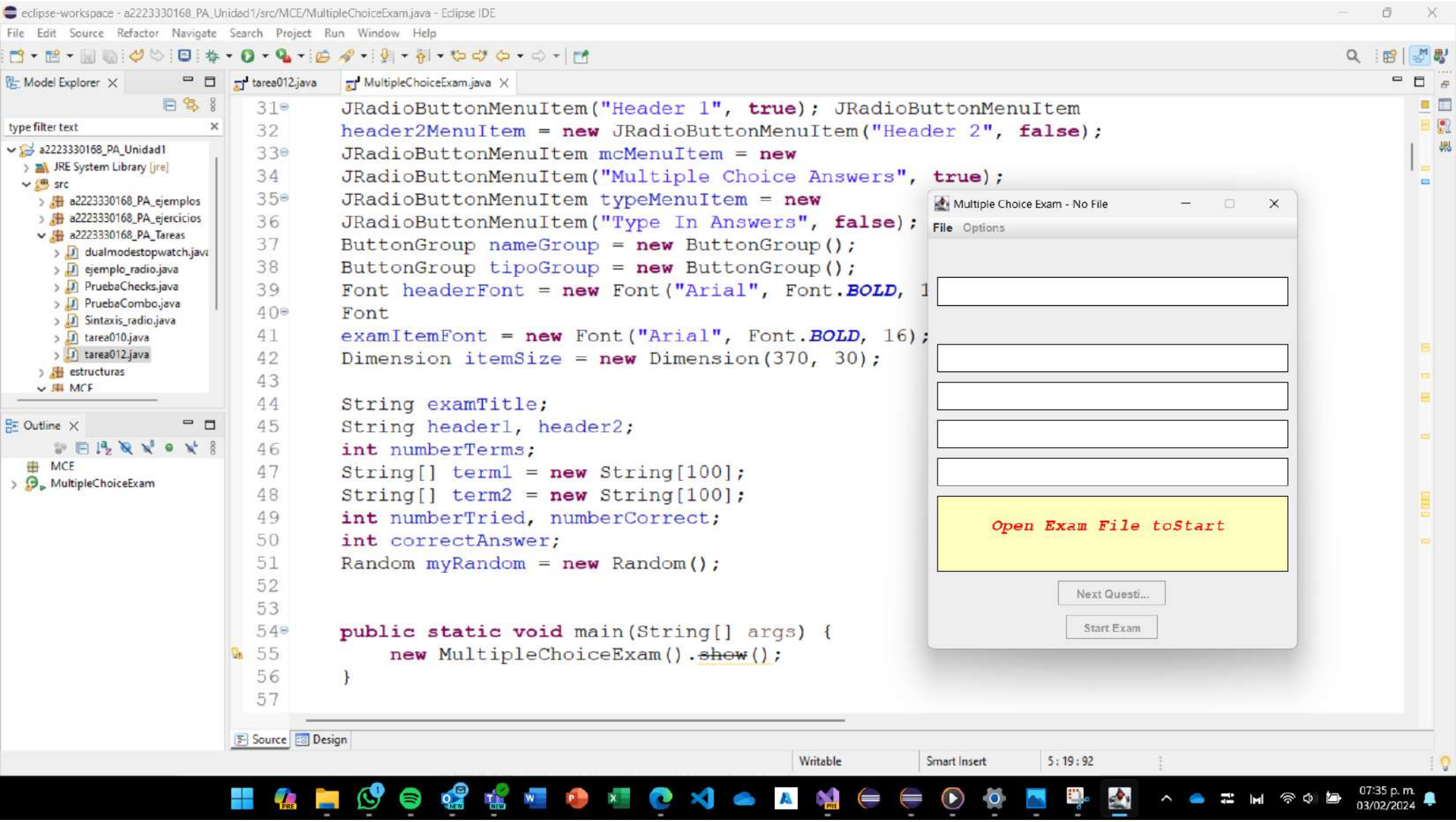
```
1  package a2223330168_PA_Tareas;
2
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import java.util. Random;
7  import java.text.*;
8     public class tarea012 extends JFrame {
9        JLabel triedLabel = new JLabel();
10       JTextField triedTextField = new JTextField();
11       JLabel correctLabel = new JLabel();
12       JTextField correctTextField = new JTextField();
13       JLabel problemLabel = new JLabel();
14       JLabel dividerLabel = new JLabel();
15       JPanel typePanel = new JPanel();
16       JCheckBox[] typeCheckBox = new JCheckBox[4];
17       JPanel factorPanel = new JPanel();
18       ButtonGroup factorButtonGroup = new ButtonGroup();
19       JRadioButton[] factorRadioButton = new JRadioButton[11]
20       JPanel timerPanel = new JPanel();
21       ButtonGroup timerButtonGroup = new ButtonGroup();
22       JRadioButton[] timerRadioButton = new JRadioButton[3];
23       JTextField timerTextField = new JTextField();
24       JScrollBar timerScrollBar = new JScrollBar();
25       JButton startButton = new JButton();
26       JButton exitButton = new JButton();
27       Timer problemsTimer;
28       Font myFont = new Font("Arial", Font.PLAIN, 18);
29       Color lightBlue = new Color (192, 192, 255);
30       Random myRandom = new Random();
31       int numberTried, numberCorrect;
32       int correctAnswer, numberDigits;
33       String problem;
34       String yourAnswer;
35       int digitNumber;
36       int problemTime;
```

Source | Design



Writable | Smart Insert | 17 : 41 : 603

07:33 p. m.
03/02/2024


eclipse-workspace - a2223330168_PA_Unidad1/src/MCE/MultipleChoiceExam.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Model Explorer
type filter text
a2223330168_PA_Unidad1
JRE System Library [jre]
src
a2223330168_PA_ejemplos
a2223330168_PA_ejercicios
a2223330168_PA_Tareas
dualmodestopwatch.java
ejemplo_radio.java
PruebaChecks.java
PruebaCombo.java
Sintaxis_radio.java
tarea010.java
tarea012.java
estructuras
MCE
Outline
MCE
MultipleChoiceExam
tarea012.java
MultipleChoiceExam.java
31 JRadioButtonMenuItem("Header 1", true); JRadioButtonMenuItem
32 header2MenuItem = new JRadioButtonMenuItem("Header 2", false);
33 JRadioButtonMenuItem mcMenuItem = new
34 JRadioButtonMenuItem("Multiple Choice Answers", true);
35 JRadioButtonMenuItem typeMenuItem = new
36 JRadioButtonMenuItem("Type In Answers", false);
37 ButtonGroup nameGroup = new ButtonGroup();
38 ButtonGroup tipoGroup = new ButtonGroup();
39 Font headerFont = new Font("Arial", Font.BOLD, 1
40 Font
41 examItemFont = new Font("Arial", Font.BOLD, 16);
42 Dimension itemSize = new Dimension(370, 30);
43
44 String examTitle;
45 String header1, header2;
46 int numberTerms;
47 String[] term1 = new String[100];
48 String[] term2 = new String[100];
49 int numberTried, numberCorrect;
50 int correctAnswer;
51 Random myRandom = new Random();
52
53
54 public static void main(String[] args) {
55 new MultipleChoiceExam().show();
56 }
57
Source
Design
Writable
Smart Insert
5 : 19 : 92
Multiple Choice Exam - No File
File Options
Open Exam File toStart
Next Questi...
Start Exam
07:35 p. m.
03/02/2024

```java
package a2223330168_PA_Tareas;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util. Random;
import java.text.*;
   public class tarea012 extends JFrame {
      JLabel triedLabel = new JLabel();
      JTextField triedTextField = new JTextField();
      JLabel correctLabel = new JLabel();
      JTextField correctTextField = new JTextField();
      JLabel problemLabel = new JLabel();
      JLabel dividerLabel = new JLabel();
      JPanel typePanel = new JPanel();
      JCheckBox[] typeCheckBox = new JCheckBox[4];
      JPanel factorPanel = new JPanel();
      ButtonGroup factorButtonGroup = new ButtonGroup
   ();
      JRadioButton[] factorRadioButton = new
   JRadioButton[11];
      JPanel timerPanel = new JPanel();
      ButtonGroup timerButtonGroup = new ButtonGroup
   ();
      JRadioButton[] timerRadioButton = new
   JRadioButton[3];
      JTextField timerTextField = new JTextField();
      JScrollBar timerScrollBar = new JScrollBar();
      JButton startButton = new JButton();
      JButton exitButton = new JButton();
      Timer problemsTimer;
      Font myFont = new Font("Arial", Font.PLAIN, 18);
      Color lightBlue = new Color (192, 192, 255);
      Random myRandom = new Random();
      int numberTried, numberCorrect;
```

```
32      int correctAnswer, numberDigits;
33      String problem;
34      String yourAnswer;
35      int digitNumber;
36      int problemTime;
37
38      public static void main(String args[]) {
39      // create frame
40      new tarea012().show();}
41
42      public tarea012(){
43      // frame constructor
44      setTitle("Flash Card Math");
45      getContentPane().setBackground(new Color (255,
  255, 192));
46      setResizable(false);
47      addWindowListener(new WindowAdapter() {
48          public void windowClosing(WindowEvent evt) {
49              exitForm(evt);
50              }
51});
52
53      getContentPane().setLayout(new GridBagLayout());
54      GridBagConstraints gridConstraints;
55      triedLabel.setText("Tried:");
56      triedLabel.setFont(myFont);
57      gridConstraints = new GridBagConstraints();
58      gridConstraints.gridx= 0;
59      gridConstraints.gridy = 0;
60      gridConstraints.anchor =
  GridBagConstraints.WEST;
61      gridConstraints.insets = new Insets(10, 10, 0,
  10);
62      getContentPane().add
  (triedLabel,gridConstraints);
```

```
 63      triedTextField.setText("0");
 64      triedTextField.setPreferredSize(new Dimension
   (90,30));
 65      triedTextField.setEditable(false);
 66      triedTextField.setBackground(Color.RED);
 67      triedTextField.setForeground(Color.YELLOW);
 68      triedTextField.setHorizontalAlignment
   (SwingConstants.CENTER);
 69      triedTextField.setFont(myFont);
 70      gridConstraints = new GridBagConstraints();
 71      gridConstraints.gridx = 1;
 72      gridConstraints.gridy = 0;
 73      gridConstraints.insets = new Insets(10, 0, 0,
   0);getContentPane().add
   (triedTextField,gridConstraints);
 74      correctLabel.setText("Correct:");
 75      correctLabel.setFont(myFont);
 76      gridConstraints = new GridBagConstraints();
 77      gridConstraints.gridx = 2;
 78      gridConstraints.gridy = 0;
 79      gridConstraints.anchor =
   GridBagConstraints.EAST;
 80      gridConstraints.insets = new Insets(10, 10, 0,
   10);
 81      getContentPane().add
   (correctLabel,gridConstraints);
 82      correctTextField.setText("0");
 83      correctTextField.setPreferredSize(new Dimension
   (90,30));
 84      correctTextField.setEditable(false);
 85      correctTextField.setBackground(Color.RED);
 86      correctTextField.setForeground(Color.YELLOW);
 87      correctTextField.setHorizontalAlignment
   (SwingConstants.CENTER);
 88      correctTextField.setFont(myFont);
```

```
 89      gridConstraints = new GridBagConstraints();
 90      gridConstraints.gridx = 3;
 91      gridConstraints.gridy = 0;
 92      gridConstraints.insets = new Insets(10, 0, 0,
   0);
 93      getContentPane().add(correctTextField,
   gridConstraints);
 94      problemLabel.setText("");
 95      problemLabel.setBorder
   (BorderFactory.createLineBorder(Color.BLACK));
 96      problemLabel.setPreferredSize(new Dimension(450,
   100));
 97      problemLabel.setBackground(Color.WHITE);
 98      problemLabel.setOpaque(true);
 99      problemLabel.setFont(new Font("Comic Sans MS",
   Font.PLAIN,48));
100      problemLabel.setHorizontalAlignment
   (SwingConstants.CENTER);
101      gridConstraints = new GridBagConstraints();
102      gridConstraints.gridx = 0;
103      gridConstraints.gridy = 1;
104      gridConstraints.gridwidth = 5;
105      gridConstraints.insets = new Insets(10, 10, 0,
   10);
106      getContentPane().add(problemLabel,
   gridConstraints);
107      problemLabel.addKeyListener(new KeyAdapter() {
108
109          public void keyPressed(KeyEvent e) {
110              problemLabelKeyPressed(e);}
111          });
112      dividerLabel.setPreferredSize(new Dimension(450,
   10));
113      dividerLabel.setBackground(Color.RED);
114      dividerLabel.setOpaque(true);
```

```
115     gridConstraints = new GridBagConstraints();
116     gridConstraints.gridx = 0;
117     gridConstraints.gridy = 2;
118     gridConstraints.gridwidth = 5;
119     gridConstraints.insets = new Insets(10, 10, 10,
    10);
120     getContentPane().add(dividerLabel,
    gridConstraints);
121     UIManager.put("TitledBorder.font", new Font
    ("Arial", Font.BOLD, 14));
122     typePanel.setPreferredSize(new Dimension(130,
    130));
123     typePanel.setBorder
    (BorderFactory.createTitledBorder("Type:"));
124     typePanel.setBackground(lightBlue);
125     typePanel.setLayout(new GridBagLayout());
    gridConstraints = new GridBagConstraints();
126
127     gridConstraints.gridx = 0;
128     gridConstraints.gridy = 3;
129     gridConstraints.gridwidth = 2;
130     gridConstraints.anchor =
    GridBagConstraints.NORTH;getContentPane().add
    (typePanel, gridConstraints); for (int i = 0; i <
    4;i++) {
131     typeCheckBox[i] = new JCheckBox();
132     typeCheckBox[i].setBackground(lightBlue);
133     gridConstraints = new GridBagConstraints();
134     gridConstraints.gridx = 0;
135     gridConstraints.gridy = i;
136     gridConstraints.anchor = GridBagConstraints.
    WEST;
137     typePanel.add(typeCheckBox[i],gridConstraints);
138     typeCheckBox[i].addActionListener(new
    ActionListener() {
```

```
139
140         public void actionPerformed(ActionEvent e) {
141       typeCheckBoxActionPerformed(e);}
142     });
143}
144     typeCheckBox[0].setText("Addition");
145     typeCheckBox[1].setText("Subtraction");
146     typeCheckBox[2].setText("Multiplication");
147     typeCheckBox[3].setText
   ("Division");typeCheckBox[0].setSelected(true);
148     factorPanel.setPreferredSize(new Dimension(130,
   130));
149     factorPanel.setBorder
   (BorderFactory.createTitledBorder("Factor:"));
150     factorPanel.setBackground(lightBlue);
151     factorPanel.setLayout(new GridBagLayout());
152     gridConstraints = new GridBagConstraints();
153     gridConstraints.gridx = 2;
154     gridConstraints.gridy = 3;
155     gridConstraints.gridwidth = 2;
156     gridConstraints.anchor =
   GridBagConstraints.NORTH;
157     getContentPane().add(factorPanel,
   gridConstraints); int x = 2;
158     int y = 0;
159     for (int i = 0; i < 11; i++)
160{
161
162    factorRadioButton[i] = new JRadioButton();
163    factorRadioButton[i].setText(String.valueOf(i));
164    factorRadioButton[i].setBackground(lightBlue);
165    factorButtonGroup.add(factorRadioButton[i]);
166    gridConstraints = new GridBagConstraints();
167    if (i < 10) {
168    gridConstraints.gridx = x;
```

```
169     gridConstraints.gridy = y;
170     }
171     else {
172     gridConstraints.gridx = 0;
173     gridConstraints.gridy = 0;
174     gridConstraints.gridwidth = 2;
175     }
176     gridConstraints.anchor = GridBagConstraints.WEST;
177     factorPanel.add(factorRadioButton[i],
    gridConstraints);
178     factorRadioButton[i].addActionListener(new
    ActionListener() {
179     public void actionPerformed(ActionEvent e) {
180     factorRadioButtonActionPerformed(e); }
181     });
182     x++;
183     if(x > 2)
184     {
185     x = 0;
186     y++;
187
188     }
189     }
190
191      factorRadioButton[10].setText("Random");
192      factorRadioButton[10].setSelected(true);
193      timerPanel.setPreferredSize(new Dimension(130,
    130));
194      timerPanel.setBorder
    (BorderFactory.createTitledBorder("Timer:"));
195      timerPanel.setBackground(lightBlue);
196      timerPanel.setLayout(new GridBagLayout());
197      gridConstraints = new GridBagConstraints();
198      gridConstraints.gridx = 4;
199      gridConstraints.gridy = 3;
```

```
200     gridConstraints.insets = new Insets(0, 0, 0,
    10);
201     gridConstraints.anchor =
    GridBagConstraints.NORTH;
202     getContentPane().add(timerPanel,
    gridConstraints);
203     for (int i = 0; i < 3; i++)
204{
205         timerRadioButton[i] = new JRadioButton();
206         timerRadioButton[i].setBackground
    (lightBlue);
207         timerButtonGroup.add(timerRadioButton[i]);
208         gridConstraints = new GridBagConstraints();
209         gridConstraints.gridx = 0;
210         gridConstraints.gridy = i;
211         gridConstraints.gridwidth = 2;
212         gridConstraints.anchor = GridBagConstraints.
    WEST;
213         timerPanel.add
    (timerRadioButton[i],gridConstraints);
214         timerRadioButton[i].addActionListener(new
    ActionListener() {
215
216             public void actionPerformed(ActionEvent
    e) {
217                 timerRadioButtonActionPerformed(e);
    }
218             });
219}
220
221     timerRadioButton[0].setText("Off");
222     timerRadioButton[1].setText("On-Count Up");
223     timerRadioButton[2].setText("On-CountDown");
224     timerRadioButton[0].setSelected(true);
225     timerTextField.setText("Off");
```

```
226     timerTextField.setPreferredSize(new Dimension
    (90,25));
227     timerTextField.setEditable(false);
228     timerTextField.setBackground(Color.WHITE);
229     timerTextField.setForeground(Color.RED);
230     timerTextField.setHorizontalAlignment
    (SwingConstants.CENTER);
231     timerTextField.setFont(myFont);
232     gridConstraints = new GridBagConstraints();
233     gridConstraints.gridx = 0;
234     gridConstraints.gridy = 3;
235     gridConstraints.anchor = GridBagConstraints.
    WEST;
236     gridConstraints.insets = new Insets(5, 0, 0, 0);
237     timerPanel.add(timerTextField,gridConstraints);
238     timerScrollBar.setPreferredSize(new Dimension
    (20, 25));
239     timerScrollBar.setMinimum(1);
240     timerScrollBar.setMaximum(60);
241     timerScrollBar.setValue(1);
242     timerScrollBar.setBlockIncrement(1);
243     timerScrollBar.setUnitIncrement(1);
244     timerScrollBar.setOrientation
    (JScrollBar.VERTICAL);
245     timerScrollBar.setEnabled(false);
246     gridConstraints = new GridBagConstraints();
247     gridConstraints.gridx = 1;
248     gridConstraints.gridy = 3;
249     gridConstraints.anchor =
    GridBagConstraints.WEST;
250     gridConstraints.insets = new Insets(5, 0, 0, 0);
251     timerPanel.add(timerScrollBar, gridConstraints);
252     timerScrollBar.addAdjustmentListener(new
    AdjustmentListener() {
253
```

```
254         public void adjustmentValueChanged
    (AdjustmentEvent e) {
255             timerScrollBarAdjustmentValueChanged(e);
    }
256           });
257     startButton.setText("Start Practice");
258     gridConstraints = new GridBagConstraints();
259     gridConstraints.gridx = 0;
260     gridConstraints.gridy = 4;
261     gridConstraints.gridwidth = 2;
262     gridConstraints.insets = new Insets(10, 0, 10,
    0);
263     getContentPane().add(startButton,
    gridConstraints);
264     startButton.addActionListener(new ActionListener
    () {
265
266         public void actionPerformed(ActionEvent e) {
267             startButtonActionPerformed(e);
268             }
269         });
270     exitButton.setText("Exit");
271     gridConstraints = new GridBagConstraints();
272     gridConstraints.gridx = 2;
273     gridConstraints.gridy = 4;
274     gridConstraints.gridwidth = 2;
275     gridConstraints.insets = new Insets(10, 0, 10,
    0);
276     getContentPane().add(exitButton,
    gridConstraints);
277     exitButton.addActionListener(new ActionListener
    () {
278         public void actionPerformed(ActionEvent e) {
279exitButtonActionPerformed(e);
280}
```

```
281 });
282 problemsTimer = new Timer (1000, new ActionListener()
    {
283 public void actionPerformed(ActionEvent e) {
284 problemsTimerActionPerformed(e);
285 }
286 });
287 pack();
288 Dimension screenSize =
    Toolkit.getDefaultToolkit().getScreenSize();
289 setBounds((int)((screenSize.width - getWidth())),
    (int) ((screenSize.height - getHeight())), getWidth
    (), getHeight());}
290 private void exitForm(WindowEvent evt) {
291 System.exit(0);
292 }
293 private void typeCheckBoxActionPerformed(ActionEvent
    e) {
294 int numberChecks;
295 int clickedBox = 0;
296 // determine which box was clicked
297 String s = e.getActionCommand();
298 if (s.equals("Addition"))
299 clickedBox = 0;
300 else if (s.equals("Subtraction")) clickedBox = 1;
301 else if (s.equals("Multiplication")) clickedBox = 2;
302 else if (s.equals("Division"))
303 clickedBox = 3;
304 // determine how many boxes are checked
305 numberChecks = 0;
306 if (typeCheckBox[0].isSelected()) numberChecks++;
307 if (typeCheckBox[1].isSelected()) numberChecks++;
308 if (typeCheckBox[2].isSelected()) numberChecks++;
309 if (typeCheckBox[3].isSelected()) {
310     numberChecks++;
```

```
311 // make sure zero not selected factor
312     if(factorRadioButton[0].isSelected())
313         factorRadioButton[1].doClick();
314         factorRadioButton[0].setEnabled(false); }
315 else
316 {
317 factorRadioButton[0].setEnabled(true); }
318 // if all boxes unchecked, recheck last clicked box
319 if (numberChecks == 0)
320     typeCheckBox[clickedBox].setSelected(true);
321     problemLabel.requestFocus();
322     }
323     private void factorRadioButtonActionPerformed
    (ActionEvent e){
324     problemLabel.requestFocus();
325     }
326     private void timerRadioButtonActionPerformed
    (ActionEvent e) {
327     if (timerRadioButton[0].isSelected()) {
328     timerTextField.setText("Off");
329     timerScrollBar.setEnabled(false); }
330     else if (timerRadioButton[1].isSelected()) {
331     problemTime = 0;
332     timerTextField.setText(getTime(problemTime));
333     timerScrollBar.setEnabled(false); }
334     else if (timerRadioButton[2].isSelected()) {
335     problemTime = 30 * timerScrollBar.getValue();
336     timerTextField.setText(getTime(problemTime));
337     timerScrollBar.setEnabled(true);
338     }
339     }
340     private void timerScrollBarAdjustmentValueChanged
    (AdjustmentEvent e) {
341     timerTextField.setText(getTime(30 *
    timerScrollBar.getValue()));}
```

```
342
343     private void startButtonActionPerformed
    (ActionEvent e){
344     int score;
345     String message = "";
346     if (startButton.getText().equals("Start
    Practice")) {
347         startButton.setText("Stop Practice");
348         exitButton.setEnabled(false);
349         numberTried = 0;
350         numberCorrect = 0;
351         triedTextField.setText("0");
352         correctTextField.setText("0");
353         timerRadioButton[0].setEnabled(false);
354         timerRadioButton[1].setEnabled(false);
355         timerRadioButton[2].setEnabled(false);
356         timerScrollBar.setEnabled(false);
357         if(!timerRadioButton[01].isSelected()) {
358         if (timerRadioButton[1].isSelected())
    problemTime = 0;
359         else
360         problemTime = 30 * timerScrollBar.getValue();
361         timerTextField.setText(getTime(problemTime));
362         problemsTimer.start();
363         }
364         problemLabel.setText(getProblem());}
365     else
366         {
367         timerRadioButton[0].setEnabled(true);
368         timerRadioButton[1].setEnabled(true);
369         timerRadioButton[2].setEnabled(true);
370         if (timerRadioButton[2].isSelected())
371             timerScrollBar.setEnabled(true);
372         problemsTimer.stop();
373         startButton.setText("Start Practice");
```

```
374         exitButton.setEnabled(true);
375         problemLabel.setText("");
376         if (numberTried > 0){
377             score = (int) (100 * (double)
    (numberCorrect) / numberTried);
378             message = "Problems Tried: " +
    String.valueOf(numberTried) + "\n";
379             message += "Problems Correct: " +
    String.valueOf(numberCorrect) + " (" +
    String.valueOf(score) + " %)" + "\n";
380             if(timerRadioButton[0].isSelected()) {
381             message += "Timer Off";
382             }
383             else
384             {
385             if(timerRadioButton[2].isSelected()) {
386             problemTime = 30 *
    timerScrollBar.getValue() - problemTime; }
387             message += "Elapsed Time: " + getTime
    (problemTime) + "\n";
388             message += "Time Per Problem: " + new
    DecimalFormat("0.00").format((double)
    (problemTime)/numberTried) + " sec";
389             }
390             JOptionPane.showConfirmDialog(null,
    message, "Results",
391             JOptionPane.DEFAULT_OPTION,
    JOptionPane.INFORMATION_MESSAGE); }
392         }
393     }
394
395             private void exitButtonActionPerformed
    (ActionEvent e){
396             System.exit(0);
397             }
```

```
398             private void problemLabelKeyPressed
    (KeyEvent e) {
399             if (startButton.getText().equals("Start
    Practice")) return;
400             // only allow number keys
401             if (e.getKeyChar() >= '0' && e.getKeyChar
    () <= '9') {
402                 yourAnswer += e.getKeyChar();
403                 problemLabel.setText(problem +
    yourAnswer);
404                 if (digitNumber != numberDigits) {
405                     digitNumber++;
406             problemLabel.setText(problemLabel.getText
    () + "?"); return;
407                 }
408     else
409     {
410     numberTried++;
411     // check answer
412     if (Integer.valueOf(yourAnswer).intValue()==
    correctAnswer)
413     {
414     numberCorrect++;
415     }
416     triedTextField.setText
    (String.valueOf(numberTried));
417     correctTextField.setText
    (String.valueOf(numberCorrect));
418     problemLabel.setText(getProblem());}
419         }
420         }
421     private void problemsTimerActionPerformed
    (ActionEvent e){
422     if (timerRadioButton[1].isSelected()) {
423         problemTime++;
```

```
424     timerTextField.setText(getTime(problemTime));
425     if (problemTime >= 1800) {
426         startButton.doClick();
427         return;
428     }
429     }
430     else
431     {
432     problemTime--;
433     timerTextField.setText(getTime(problemTime));
434     if (problemTime == 0)
435     {
436     startButton.doClick();
437     return;
438     }
439     }
440     }
441
442     private String getProblem() {
443
444
445     int pType, p, number, factor;
446     p = 0;
447     do
448         {
449         pType = myRandom.nextInt(4) + 1;
450     if (pType == 1 && typeCheckBox[0].isSelected()) {
451     // Addition
452         p = pType;
453         number = myRandom.nextInt(10);
454         factor = getFactor(1);
455         correctAnswer = number + factor;
456         problem = String.valueOf(number) + " + " +
   String.valueOf(factor) + " = "; }
457         else if (pType == 2 &&
```

```
   typeCheckBox[1].isSelected()) {
458         // Subtraction
459         p = pType;
460         factor = getFactor(2);
461         correctAnswer = myRandom.nextInt(10);
462         number = correctAnswer + factor;
463         problem = String.valueOf(number) + " - " +
   String.valueOf(factor) + " ="; }
464         else if (pType == 3 &&
   typeCheckBox[2].isSelected()) {
465         // Multiplication
466         p = pType;
467         number = myRandom.nextInt(10);
468         factor = getFactor(3);
469         correctAnswer = number * factor;
470         problem = String.valueOf(number) + " × " +
   String.valueOf(factor) + " = "; }
471         else if (pType == 4 &&
   typeCheckBox[3].isSelected()) {
472         // Division
473         p = pType;
474         factor = getFactor(4);
475         correctAnswer = myRandom.nextInt(10);
476         number = correctAnswer * factor;
477         problem = String.valueOf(number) + " / " +
   String.valueOf(factor) + " ="; }
478         }
479         while (p == 0);
480             yourAnswer ="";
481             digitNumber = 1;
482             problemLabel.requestFocus();
483             if (correctAnswer < 10)
484             {
485             numberDigits = 1;
486             return (problem + "?");
```

```
487             }
488             else
489             {
490             numberDigits = 2;
491             return (problem +"??");
492             }
493     }
494             private int getFactor(int p)
495             {
496             if (factorRadioButton[10].isSelected()) {
497             //random
498             if (p == 4)
499             return (myRandom.nextInt(9) + 1); else
500                 return (myRandom.nextInt(10));
501             }
502             else
503             {
504                 for (int i = 0; i < 10; i++)
505                 {
506                 if (factorRadioButton[i].isSelected
    ())
507                     return(i);
508                 }
509                 return (0);
510             }
511             }
512             private String getTime(int s)
513             {
514                 int min, sec;
515                 String ms, ss;
516                 min = (int) (s / 60);
517                 sec = s - 60 * min;
518                 ms = String.valueOf(min);
519                 ss = String.valueOf(sec);
520                 if (sec < 10)
```

```
521                 ss = "0" + ss;
522                 return (ms + ":" + ss);
523
524 }
525 }
```

```
 1 package MCE;
 2
 3 import javax.swing.filechooser.*;
 4 import javax.swing.*;
 5 import java.awt.*;
 6 import java.awt.event.*;
 7 import java.io.*;
 8 import java.util.Random;
 9 import java.text.*;
10 import java.awt.EventQueue;
11
12 import javax.swing.border.EmptyBorder;
13
14 public class MultipleChoiceExam extends JFrame {
15
16     JLabel headGivenLabel = new JLabel();
17     JLabel givenLabel = new JLabel();
18     JLabel headAnswerLabel = new JLabel();
19     JLabel[] answerLabel = new JLabel[4];
20     JTextField answerTextField = new JTextField();
21     JTextArea commentTextArea = new JTextArea();
22     JButton nextButton = new JButton();
23     JButton startButton = new JButton();
24     // menu structure
25     JMenuBar mainMenuBar = new JMenuBar();
26     JMenu fileMenu = new JMenu("File");
27     JMenuItem openMenuItem = new JMenuItem("Open");
28     JMenuItem exitMenuItem = new JMenuItem("Exit");
29     JMenu optionsMenu = new JMenu("Options");
30     JRadioButtonMenuItem header1MenuItem = new
31     JRadioButtonMenuItem("Header 1", true);
   JRadioButtonMenuItem
32     header2MenuItem = new JRadioButtonMenuItem
   ("Header 2", false);
33     JRadioButtonMenuItem mcMenuItem = new
```

```
34      JRadioButtonMenuItem("Multiple Choice Answers",
    true);
35      JRadioButtonMenuItem typeMenuItem = new
36      JRadioButtonMenuItem("Type In Answers", false);
37      ButtonGroup nameGroup = new ButtonGroup();
38      ButtonGroup tipoGroup = new ButtonGroup();
39      Font headerFont = new Font("Arial", Font.BOLD,
    18);
40      Font
41      examItemFont = new Font("Arial", Font.BOLD, 16);
42      Dimension itemSize = new Dimension(370, 30);
43
44      String examTitle;
45      String header1, header2;
46      int numberTerms;
47      String[] term1 = new String[100];
48      String[] term2 = new String[100];
49      int numberTried, numberCorrect;
50      int correctAnswer;
51      Random myRandom = new Random();
52
53
54      public static void main(String[] args) {
55          new MultipleChoiceExam().show();
56      }
57
58      public MultipleChoiceExam() {
59          setTitle("Multiple Choice Exam - No File");
60          setResizable(false);
61          addWindowListener(new WindowAdapter()
62          {
63          public void windowClosing(WindowEvent evt)
64          {
65          exitForm(evt);
66          }
```

```
67
68          private void exitForm(WindowEvent evt) {
69              // TODO Auto-generated method stub
70
71          }
72          });
73          getContentPane().setLayout(new GridBagLayout
    ());
74          GridBagConstraints gridConstraints;
75          headGivenLabel.setPreferredSize(itemSize);
76          headGivenLabel.setFont(headerFont);
77          gridConstraints = new GridBagConstraints();
78          gridConstraints.gridx = 0;
79          gridConstraints.gridy = 0;
80          gridConstraints.insets = new Insets(10, 10,
    0, 10);
81          getContentPane().add(headGivenLabel,
    gridConstraints);
82          givenLabel.setPreferredSize(itemSize);
83          givenLabel.setFont(examItemFont);
84          givenLabel.setBorder
    (BorderFactory.createLineBorder(Color.BLACK));
85          givenLabel.setBackground(Color.WHITE);
86          givenLabel.setForeground(Color.BLUE);
87          givenLabel.setOpaque(true);
88          givenLabel.setHorizontalAlignment
    (SwingConstants.CENTER);
89          gridConstraints = new GridBagConstraints();
90          gridConstraints.gridx = 0;
91          gridConstraints.gridy = 1;
92          gridConstraints.insets = new Insets(0, 10, 0,
    10);
93          getContentPane().add(givenLabel,
    gridConstraints);
94          headAnswerLabel.setPreferredSize(itemSize);
```

```
 95          headAnswerLabel.setFont(headerFont);
 96          gridConstraints = new GridBagConstraints();
 97          gridConstraints.gridx = 0;
 98          gridConstraints.gridy = 2;
 99          gridConstraints.insets = new Insets(10, 10,
    0, 10);
100          getContentPane().add(headAnswerLabel,
    gridConstraints);
101          for (int i= 0; i < 4; i++){
102                  answerLabel[i] = new JLabel();
103                  answerLabel[i].setPreferredSize
    (itemSize);
104                  answerLabel[i].setFont(examItemFont);
105                  answerLabel[i].setBorder
    (BorderFactory.createLineBorder(Color.BLACK));
106                  answerLabel[i].setBackground
    (Color.WHITE);
107                  answerLabel[i].setForeground
    (Color.BLUE);
108                  answerLabel[i].setOpaque(true);
109                  answerLabel[i].setHorizontalAlignment
    (SwingConstants.CENTER);
110                  gridConstraints = new
    GridBagConstraints();
111                  gridConstraints.gridx = 0;
112                  gridConstraints.gridy = i + 3;
113                  gridConstraints.insets = new Insets
    (0, 10, 10, 10);
114                  getContentPane().add(answerLabel[i],
    gridConstraints);
115                  answerLabel[i].addMouseListener(new
    MouseAdapter() {
116                  public void mousePressed(MouseEvent
    e)
117                  {
```

```
118                 mousePressed(e);
119                 }
120                 });
121                 }
122                 answerTextField.setPreferredSize
    (itemSize);
123                 answerTextField.setFont
    (examItemFont);
124                 answerTextField.setBackground
    (Color.WHITE);
125                 answerTextField.setForeground
    (Color.BLUE);
126                 answerTextField.setVisible(false);
127                 gridConstraints = new
    GridBagConstraints();
128                 gridConstraints.gridx = 0;
129                 gridConstraints.gridy = 3;
130                 gridConstraints.insets = new Insets
    (0, 10, 10, 10);
131                 getContentPane().add(answerTextField,
    gridConstraints);
132                 answerTextField.addActionListener(new
    ActionListener () {
133                 public void actionPerformed
    (ActionEvent e)
134                 {
135                 actionPerformed(e);
136                 }
137                 });
138                 commentTextArea.setPreferredSize(new
    Dimension(370, 80));
139                 commentTextArea.setFont(new Font
    ("Courier New", Font.BOLD +
140                 Font.ITALIC, 18));
141                 commentTextArea.setBorder
```

```
    (BorderFactory.createLineBorder(Color.BLACK));
142                 commentTextArea.setEditable(false);
143                 commentTextArea.setBackground(new
    Color(255, 255, 192));
144                 commentTextArea.setForeground
    (Color.RED);
145                 gridConstraints = new
    GridBagConstraints();
146                 gridConstraints.gridx = 0;
147                 gridConstraints.gridy = 7;
148                 gridConstraints.insets = new Insets
    (0, 10, 10, 10);
149                 getContentPane().add(commentTextArea,
    gridConstraints);
150                 nextButton.setText("Next Question");
151                 gridConstraints = new
    GridBagConstraints();
152                 gridConstraints.gridx = 0;
153                 gridConstraints.gridy = 8;
154                 gridConstraints.insets = new Insets
    (0, 0, 10, 0);
155                 getContentPane().add(nextButton,
    gridConstraints);
156                 nextButton.addActionListener(new
    ActionListener() {
157                 public void actionPerformed
    (ActionEvent e)
158                 {
159                 actionPerformed(e);
160                 }
161                 });
162                 startButton.setText("Start Exam");
163                 gridConstraints = new
    GridBagConstraints();
164                 gridConstraints.gridx = 0;
```

```
165                 gridConstraints.gridy = 9;
166                 gridConstraints.insets = new Insets
    (0, 0, 10, 0);
167                 getContentPane().add(startButton,
    gridConstraints);
168                 startButton.addActionListener(new
    ActionListener() {
169                 public void actionPerformed
    (ActionEvent e)
170                 {
171                 actionPerformed(e);
172                 }
173                 });
174                 // build menu structure
175                 setJMenuBar(mainMenuBar);
176                 mainMenuBar.add(fileMenu);
177                 fileMenu.add(openMenuItem);
178                 fileMenu.addSeparator();
179                 fileMenu.add(exitMenuItem);
180                 mainMenuBar.add(optionsMenu);
181                 optionsMenu.add(header1MenuItem);
182                 optionsMenu.add(header2MenuItem);
183                 optionsMenu.addSeparator();
184                 optionsMenu.add(mcMenuItem);
185                 optionsMenu.add(typeMenuItem);
186                 nameGroup.add(header1MenuItem);
187                 nameGroup.add(header2MenuItem);
188                 tipoGroup.add(mcMenuItem);
189                 tipoGroup.add(typeMenuItem);
190                 openMenuItem.addActionListener(new
    ActionListener() {
191                 public void actionPerformed
    (ActionEvent e)
192                 {
193                     actionPerformed(e);
```

```
194                 }
195                 });
196                 exitMenuItem.addActionListener(new
    ActionListener() {
197                 public void actionPerformed
    (ActionEvent e)
198                 {
199                 actionPerformed(e);
200                 }
201                 });
202                 header1MenuItem.addActionListener(new
    ActionListener() {
203                 public void actionPerformed
    (ActionEvent e)
204                 {
205                 actionPerformed(e);
206                 }
207                 });
208                 header2MenuItem.addActionListener(new
    ActionListener() {
209                 public void actionPerformed
    (ActionEvent e)
210                 {
211                 actionPerformed(e);
212                 }
213                 });
214                 mcMenuItem.addActionListener(new
    ActionListener() {
215                 public void actionPerformed
    (ActionEvent e)
216                 {
217                 actionPerformed(e);
218                 }
219                 });
220                 typeMenuItem.addActionListener(new
```

```
   ActionListener() {
221                public void actionPerformed
   (ActionEvent e)
222                {
223                actionPerformed(e);
224                }
225                });
226                pack();
227                Dimension screenSize =
   Toolkit.getDefaultToolkit().getScreenSize();
228                setBounds((int) (0.5*
   (screenSize.width - getWidth())), (int) (0.5*
   (screenSize.height - getHeight())), getWidth(),
   getHeight()); // initialize form
229                startButton.setEnabled(false);
230                nextButton.setEnabled(false);
231                optionsMenu.setEnabled(false);
232                commentTextArea.setText
   (centerTextArea("Open Exam File toStart")); }
233                private void exitForm(WindowEvent
   evt)
234                {
235                System.exit(0);
236                }
237                private void answerLabelMousePressed
   (MouseEvent e) {
238                boolean correct = false;
239                int labelSelected;
240                // make sure exam has started and
   question has not been answered if
241                if(startButton.getText().equals
   ("Start Exam") || nextButton.isEnabled())
242                return;
243                // determine which label was clicked
244                // get upper left corner of clicked
```

```
    label
245                 Point p = e.getComponent
    ().getLocation();
246                 // determine index based on p
247                 for (labelSelected = 0; labelSelected
    < 20; labelSelected++) {
248                 if (p.x ==
    answerLabel[labelSelected].getX() && p.y ==
249                 answerLabel[labelSelected].getY())
    break;
250                 }
251                 // If already answered, exit
252                 numberTried++;
253                 if (header1MenuItem.isSelected())
254                 {
255                 if
    (answerLabel[labelSelected].getText().equals
    (term1[correctAnswer])) correct = true;
256                 }
257                 else
258                 {
259                 if
    (answerLabel[labelSelected].getText().equals
    (term2[correctAnswer])) correct = true;
260                 }
261                 updateScore(correct);
262                 }
263                 private void
    answerTextFieldActionPerformed(ActionEvent e) {
264                 // Check type in answer
265                 boolean correct;
266                 String ucTypedAnswer, ucAnswer;
267                 // make sure exam has started and
    question has not been answered if
268                 if(startButton.getText().equals
```

```
    ("Start Exam") || nextButton.isEnabled())
269                 return;
270                 answerTextField.setEditable(false);
271                 numberTried++;
272                 ucTypedAnswer =
    answerTextField.getText().toUpperCase(); if
273                 (header1MenuItem.isSelected())
274                 ucAnswer =
    term1[correctAnswer].toUpperCase();
275                 else
276                 ucAnswer =
    term2[correctAnswer].toUpperCase();
277                 correct = false;
278                 if (ucTypedAnswer.equals(ucAnswer) ||
279                 soundex(ucTypedAnswer).equals(soundex
    (ucAnswer))) correct = true;
280                 updateScore(correct);
281                 }
282                 private void
    nextButtonActionPerformed(ActionEvent e) {
283                 // Generate next question
284                 nextButton.setEnabled(false);
285                 nextQuestion();
286                 }
287                 private void
    startButtonActionPerformed(ActionEvent e) {
288                 String message;
289                 if (startButton.getText().equals
    ("Start Exam"))
290                 {
291                 startButton.setText("Stop Exam");
292                 nextButton.setEnabled(false);
293                 // Reset the score
294                 numberTried = 0;
295                 numberCorrect = 0;
```

```
296                 commentTextArea.setText("");
297                 fileMenu.setEnabled(false);
298                 optionsMenu.setEnabled(false);
299                 nextQuestion();
300                 }
301                 else
302                 {
303                 startButton.setText("Start Exam");
304                 nextButton.setEnabled(false);
305                 if (numberTried > 0)
306                 {
307                 message = "Questions Tried: " +
    String.valueOf(numberTried)
308                 + "\n"; message += "Questions
    Correct: " +
309                 String.valueOf(numberCorrect) + "\n
    \n"; message += "Your Score: " +
310                 new DecimalFormat("0.0").format(100.0
    * ((double) numberCorrect /
311                 numberTried)) + "%";
    JOptionPane.showConfirmDialog(null, message,
312                 examTitle + " Results",
    JOptionPane.DEFAULT_OPTION,
313                 JOptionPane.INFORMATION_MESSAGE); }
314                 givenLabel.setText("");
315                 answerLabel[0].setText("");
316                 answerLabel[1].setText("");
317                 answerLabel[2].setText("");
318                 answerLabel[3].setText("");
319                 answerTextField.setText("");
320                 commentTextArea.setText
    (centerTextArea("ChooseOptions\nClick Start Exam"));
    fileMenu.setEnabled(true);
321                 optionsMenu.setEnabled(true);
322                 }
```

```
323                 }
324                 private void
    openMenuItemActionPerformed(ActionEvent e) {
325                 String myLine;
326                 JFileChooser openChooser = new
    JFileChooser();
327                 openChooser.setDialogType
    (JFileChooser.OPEN_DIALOG);
328                 openChooser.setDialogTitle("Open Exam
    File");
329                 openChooser.addChoosableFileFilter
    (new
330                 FileNameExtensionFilter("Exam Files",
    "csv")); if
331                 (openChooser.showOpenDialog(this) ==
332                 JFileChooser.APPROVE_OPTION) {
333                 try
334                 {
335                 BufferedReader inputFile = new
    BufferedReader(new
336                 FileReader
    (openChooser.getSelectedFile())); myLine =
337                 inputFile.readLine();
338                 examTitle = parseLeft(myLine);
339                 myLine = inputFile.readLine();
340                 header1 = parseLeft(myLine);
341                 header2 = parseRight(myLine);
342                 numberTerms = 0;
343                 do
344                 {
345                 numberTerms++;
346                 myLine = inputFile.readLine();
347                 term1[numberTerms - 1] = parseLeft
    (myLine);
348                 term2[numberTerms - 1] = parseRight
```

```
(myLine);
349                 }
350                 while (inputFile.ready() &&
numberTerms < 100); if
351                 (numberTerms < 5)
352                 {
353                 JOptionPane.showConfirmDialog(null,
"Must have at least 5 entries in exam file.", "Exam
File Error",
354                 JOptionPane.DEFAULT_OPTION,
JOptionPane.ERROR_MESSAGE);
355                 return;
356                 }
357                 inputFile.close();
358                 // establish frame title
359                 this.setTitle("Multiple Choice Exam -
" + examTitle); // set up menu items
360                 header1MenuItem.setText(header1 + ",
Given " + header2);
361                 header2MenuItem.setText(header2 + ",
Given " + header1); if
362                 (header1MenuItem.isSelected())
363                 {
364                 headGivenLabel.setText(header2);
365                 headAnswerLabel.setText(header1);
366                 }
367                 else
368                 {
369                 headGivenLabel.setText(header1);
370                 headAnswerLabel.setText(header2);
371                 }
372                 startButton.setEnabled(true);
373                 optionsMenu.setEnabled(true);
374                 commentTextArea.setText
(centerTextArea("File Loaded,Choose Options\nClick
```

```
   Start Exam")); }
375                 catch (Exception ex)
376                 {
377                 JOptionPane.showConfirmDialog(null,
   "Error reading in input file - make sure file is
   correct format.", "Multiple Choice Exam File Error",
   JOptionPane.DEFAULT_OPTION,
378                 JOptionPane.ERROR_MESSAGE); return;
379                 }
380                 }
381                 }
382                 private void
   exitMenuItemActionPerformed(ActionEvent e) {
383                 System.exit(0);
384                 }
385                 private void
   header1MenuItemActionPerformed(ActionEvent e) {
386                 // Set up for naming header1, given
   header2
387                 headGivenLabel.setText(header2);
388                 headAnswerLabel.setText(header1);
389                 }
390                 private void
   header2MenuItemActionPerformed(ActionEvent e) {
391                 // Set up for naming header2, given
   header1
392                 headGivenLabel.setText(header1);
393                 headAnswerLabel.setText(header2);
394                 }
395                 private void
   mcMenuItemActionPerformed(ActionEvent e) {
396                 answerLabel[0].setVisible(true);
397                 answerLabel[1].setVisible(true);
398                 answerLabel[2].setVisible(true);
399                 answerLabel[3].setVisible(true);
```

```
400                 answerTextField.setVisible(false);
401                 }
402                 private void
    typeMenuItemActionPerformed(ActionEvent e) {
403                 answerLabel[0].setVisible(false);
404                 answerLabel[1].setVisible(false);
405                 answerLabel[2].setVisible(false);
406                 answerLabel[3].setVisible(false);
407                 answerTextField.setVisible(true);
408                 }
409                 private String parseLeft(String s)
410                 {
411                 int cl;
412                 // find comma
413                 cl = s.indexOf(",");
414                 return (s.substring(0, cl));
415                 }
416                 private String parseRight(String s)
417                 {
418                 int cl;
419                 // find comma
420                 cl = s.indexOf(",");
421                 return (s.substring(cl + 1));
422                 }
423                 private String centerTextArea(String
    s)
424                 {
425                 // centers up to two lines in text
    area
426                 int charsPerLine = 33;
427                 String sOut = "";
428                 int j = s.indexOf("\n");
429                 int nSpaces;
430                 if (j == -1)
431                 {
```

```
432                 // single line
433                 sOut = "\n" + spacePadding((int)
    ((charsPerLine - s.length()) / 2))
434                 + s; }
435                 else
436                 {
437                 // first line
438                 String l = s.substring(0, j);
439                 sOut = "\n" + spacePadding((int)
    ((charsPerLine - l.length()) / 2))
440                 + l; // second line
441                 l = s.substring(j + 1);
442                 sOut += "\n" + spacePadding((int)
    ((charsPerLine - l.length()) / 2))
443                 + l ; }
444                 return(sOut);
445                 }
446                 private String spacePadding(int n)
447                 {
448                 String s = "";
449                 if (n != 0)
450                 for (int i = 0; i < n; i++)
451                 s += " ";
452                 return(s);
453                 }
454                 private void nextQuestion()
455                 {
456                 boolean[] termUsed = new
    boolean[numberTerms];
457                 int[] index = new int[4];
458                 int j;
459                 commentTextArea.setText("");
460                 // Generate the next question based
    on selected options
461                 correctAnswer = myRandom.nextInt
```

```
    (numberTerms);
462                 if (header1MenuItem.isSelected())
463                 {
464                 givenLabel.setText
    (term2[correctAnswer]);
465                 }
466                 else
467                 {
468                 givenLabel.setText
    (term1[correctAnswer]);
469                 }
470                 if (mcMenuItem.isSelected())
471                 {
472                 // Multiple choice answers
473                 for (int i = 0; i < numberTerms; i++)
474                 {
475                 termUsed[i] = false;
476                 }
477
478                 // Pick four random possiblities
479                 for (int i = 0; i < 4; i++)
480                 {
481                 do
482                 {
483                 j = myRandom.nextInt(numberTerms);
484                 }
485                 while (termUsed[j] || j ==
    correctAnswer);
486                 termUsed[j] = true;
487                 index[i] = j;
488                 }
489                 // Replace one with correct answer
490                 index[myRandom.nextInt(4)] =
    correctAnswer;
491                 // Display multiple choice answers in
```

```
    label boxes if
492                 if(header1MenuItem.isSelected())
493                 {
494                 answerLabel[0].setText
    (term1[index[0]]);
495                 answerLabel[1].setText
    (term1[index[1]]);
496                 answerLabel[2].setText
    (term1[index[2]]);
497                 answerLabel[3].setText
    (term1[index[3]]);
498                 }
499                 else
500                 {
501                 answerLabel[0].setText
    (term2[index[0]]);
502                 answerLabel[1].setText
    (term2[index[1]]);
503                 answerLabel[2].setText
    (term2[index[2]]);
504                 answerLabel[3].setText
    (term2[index[3]]);
505                 }
506                 }
507                 else
508                 {
509                 // Type-in answers
510                 answerTextField.setEditable(true);
511                 answerTextField.setText("");
512                 answerTextField.requestFocus();
513                 }
514                 }
515                 private void updateScore(boolean
    correct)
516                 {
```

```
517                 // Check if answer is correct
518                 if (correct)
519                 {
520                 numberCorrect++;
521                 commentTextArea.setText
    (centerTextArea("Correct!")); }
522                 else
523                 commentTextArea.setText
    (centerTextArea("Sorry ... CorrectAnswer Shown")); //
    Display correct answer
524                 if (mcMenuItem.isSelected())
525                 {
526                 if (header1MenuItem.isSelected())
527                 answerLabel[0].setText
    (term1[correctAnswer]);
528                 else
529                 answerLabel[0].setText
    (term2[correctAnswer]);
530                 answerLabel[1].setText("");
531                 answerLabel[2].setText("");
532                 answerLabel[3].setText("");
533                 }
534                 else
535                 {
536                 if (header1MenuItem.isSelected())
537                 answerTextField.setText
    (term1[correctAnswer]);
538                 else
539                 answerTextField.setText
    (term2[correctAnswer]);
540                 }
541                 startButton.setEnabled(true);
542                 nextButton.setEnabled(true);
543                 nextButton.requestFocus();
544                 }
```

```
545                 public String soundex(String w)
546                 {
547                 // Generates Soundex code for W based
    on Unicode value // Allowsanswers whose spelling is
    close, but not exact
548                     String wTemp, s = "";
549                 int l;
550                 int wPrev, wSnd, cIndex;
551                 // Load soundex function array
552                 int[] wSound = {0, 1, 2, 3, 0, 1, 2,
    0, 0, 2, 2, 4, 5, 5, 0, 1, 2, 6, 2, 3, 0, 1, 0,
553                 2, 0, 2}; wTemp = w.toUpperCase();
554                 l = w.length();
555                 if (l != 0)
556                 {
557                 s = String.valueOf(w.charAt(0));
558                 wPrev = 0;
559                 if (l > 1)
560                 {
561                 for (int i = 1; i < l; i++)
562                 {
563                 cIndex = (int) wTemp.charAt(i) - 65;
564                 if (cIndex >= 0 && cIndex <= 25)
565                 {
566                 wSnd = wSound[cIndex] + 48;
567                 if (wSnd != 48 && wSnd != wPrev)
568                 {
569                 s += String.valueOf((char) wSnd);
570                 }
571                 wPrev = wSnd;
572                 }
573                 }
574                 }
575                 else s=" ";{
576                     return(s);
```

```
577                     }
578                     }
579                     return s;
580
581                 }
582   }
583
```