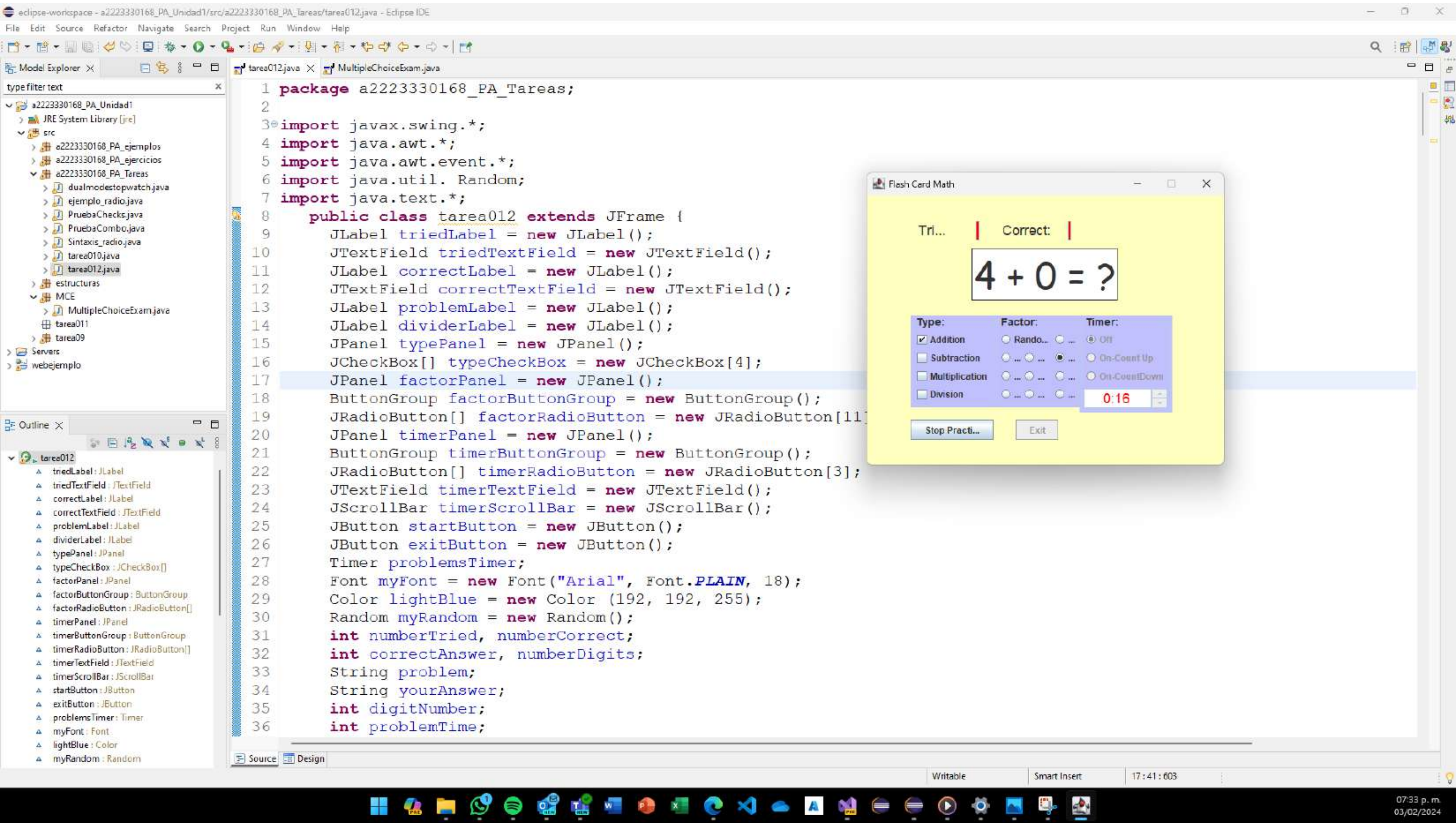


```java
package a2223330168_PA_Tareas;

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;
    int correctAnswer, numberDigits;
    String problem;
    String yourAnswer;
    int digitNumber;
    int problemTime;
```

Flash Card Math

Tri... Correct:

4 + 0 = ?

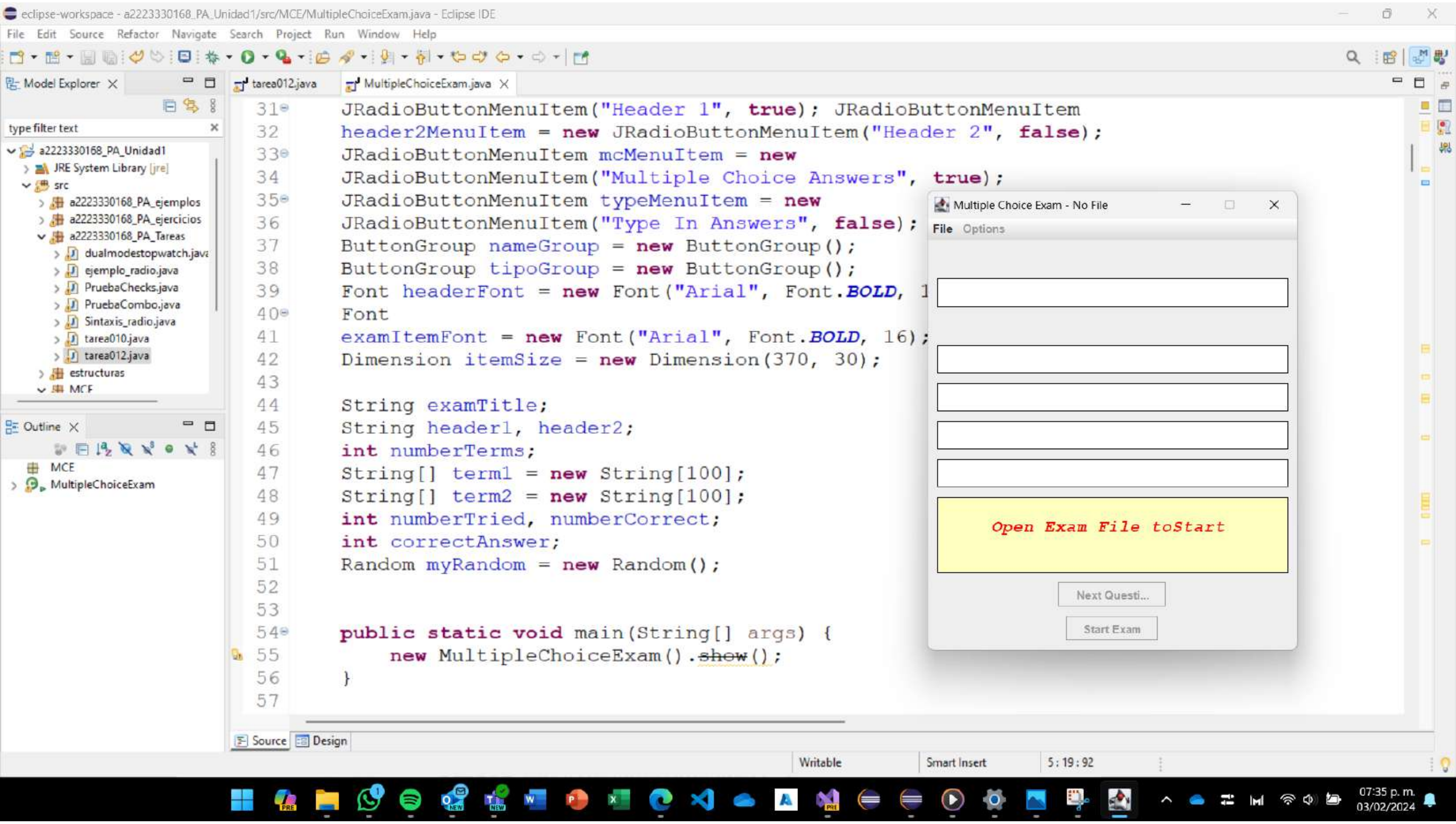
Type: [✓ Addition] Subtraction Multiplication Division

Factor: Rando... ... ... [● ...] ... ...

Timer: [● Off] On-Count Up On-CountDown

0:16

Stop Practi... Exit



```
    JRadioButtonMenuItem("Header 1", true); JRadioButtonMenuItem
    header2MenuItem = new JRadioButtonMenuItem("Header 2", false);
    JRadioButtonMenuItem mcMenuItem = new
    JRadioButtonMenuItem("Multiple Choice Answers", true);
    JRadioButtonMenuItem typeMenuItem = new
    JRadioButtonMenuItem("Type In Answers", false);
    ButtonGroup nameGroup = new ButtonGroup();
    ButtonGroup tipoGroup = new ButtonGroup();
    Font headerFont = new Font("Arial", Font.BOLD, 1
    Font
    examItemFont = new Font("Arial", Font.BOLD, 16);
    Dimension itemSize = new Dimension(370, 30);

    String examTitle;
    String header1, header2;
    int numberTerms;
    String[] term1 = new String[100];
    String[] term2 = new String[100];
    int numberTried, numberCorrect;
    int correctAnswer;
    Random myRandom = new Random();


    public static void main(String[] args) {
        new MultipleChoiceExam().~~show~~();
    }

```

```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;
```

```java
    int correctAnswer, numberDigits;
    String problem;
    String yourAnswer;
    int digitNumber;
    int problemTime;

    public static void main(String args[]) {
    // create frame
    new tarea012().~~show~~();}

    public tarea012(){
    // frame constructor
    setTitle("Flash Card Math");
    getContentPane().setBackground(new Color (255, 255, 192));
    setResizable(false);
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent evt) {
            exitForm(evt);
            }
});

    getContentPane().setLayout(new GridBagLayout());
    GridBagConstraints gridConstraints;
    triedLabel.setText("Tried:");
    triedLabel.setFont(myFont);
    gridConstraints = new GridBagConstraints();
    gridConstraints.gridx= 0;
    gridConstraints.gridy = 0;
    gridConstraints.anchor = GridBagConstraints.WEST;
    gridConstraints.insets = new Insets(10, 10, 0, 10);
    getContentPane().add (triedLabel,gridConstraints);
    triedTextField.setText("0");
    triedTextField.setPreferredSize(new Dimension
  (90,30));
    triedTextField.setEditable(false);
    triedTextField.setBackground(Color.RED);
    triedTextField.setForeground(Color.YELLOW);
    triedTextField.setHorizontalAlignment
  (SwingConstants.CENTER);
    triedTextField.setFont(myFont);
    gridConstraints = new GridBagConstraints();
    gridConstraints.gridx = 1;
    gridConstraints.gridy = 0;
    gridConstraints.insets = new Insets(10, 0, 0,
  0);getContentPane().add
  (triedTextField,gridConstraints);
    correctLabel.setText("Correct:");
    correctLabel.setFont(myFont);
    gridConstraints = new GridBagConstraints();
    gridConstraints.gridx = 2;
    gridConstraints.gridy = 0;
    gridConstraints.anchor =
  GridBagConstraints.EAST;
    gridConstraints.insets = new Insets(10, 10, 0,
  10);
    getContentPane().add
  (correctLabel,gridConstraints);
    correctTextField.setText("0");
    correctTextField.setPreferredSize(new Dimension
  (90,30));
    correctTextField.setEditable(false);
    correctTextField.setBackground(Color.RED);
    correctTextField.setForeground(Color.YELLOW);
    correctTextField.setHorizontalAlignment
  (SwingConstants.CENTER);
    correctTextField.setFont(myFont);
```

```java
    gridConstraints = new GridBagConstraints();
    gridConstraints.gridx = 3;
    gridConstraints.gridy = 0;
    gridConstraints.insets = new Insets(10, 0, 0, 0);
    getContentPane().add(correctTextField, gridConstraints);
    problemLabel.setText("");
    problemLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    problemLabel.setPreferredSize(new Dimension(450, 100));
    problemLabel.setBackground(Color.WHITE);
    problemLabel.setOpaque(true);
    problemLabel.setFont(new Font("Comic Sans MS", Font.PLAIN,48));
    problemLabel.setHorizontalAlignment(SwingConstants.CENTER);
    gridConstraints = new GridBagConstraints();
    gridConstraints.gridx = 0;
    gridConstraints.gridy = 1;
    gridConstraints.gridwidth = 5;
    gridConstraints.insets = new Insets(10, 10, 0, 10);
    getContentPane().add(problemLabel, gridConstraints);
    problemLabel.addKeyListener(new KeyAdapter() {

        public void keyPressed(KeyEvent e) {
            problemLabelKeyPressed(e);}
        });
    dividerLabel.setPreferredSize(new Dimension(450, 10));
    dividerLabel.setBackground(Color.RED);
    dividerLabel.setOpaque(true);
    gridConstraints = new GridBagConstraints();
    gridConstraints.gridx = 0;
    gridConstraints.gridy = 2;
    gridConstraints.gridwidth = 5;
    gridConstraints.insets = new Insets(10, 10, 10, 10);
    getContentPane().add(dividerLabel, gridConstraints);
    UIManager.put("TitledBorder.font", new Font("Arial", Font.BOLD, 14));
    typePanel.setPreferredSize(new Dimension(130, 130));
    typePanel.setBorder(BorderFactory.createTitledBorder("Type:"));
    typePanel.setBackground(lightBlue);
    typePanel.setLayout(new GridBagLayout()); gridConstraints = new GridBagConstraints();

    gridConstraints.gridx = 0;
    gridConstraints.gridy = 3;
    gridConstraints.gridwidth = 2;
    gridConstraints.anchor = GridBagConstraints.NORTH;getContentPane().add(typePanel, gridConstraints); for (int i = 0; i < 4;i++) {
    typeCheckBox[i] = new JCheckBox();
    typeCheckBox[i].setBackground(lightBlue);
    gridConstraints = new GridBagConstraints();
    gridConstraints.gridx = 0;
    gridConstraints.gridy = i;
    gridConstraints.anchor = GridBagConstraints.WEST;
    typePanel.add(typeCheckBox[i],gridConstraints);
    typeCheckBox[i].addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
         typeCheckBoxActionPerformed(e);}
     });
}
        typeCheckBox[0].setText("Addition");
        typeCheckBox[1].setText("Subtraction");
        typeCheckBox[2].setText("Multiplication");
        typeCheckBox[3].setText
 ("Division");typeCheckBox[0].setSelected(true);
        factorPanel.setPreferredSize(new Dimension(130,
 130));
        factorPanel.setBorder
 (BorderFactory.createTitledBorder("Factor:"));
        factorPanel.setBackground(lightBlue);
        factorPanel.setLayout(new GridBagLayout());
        gridConstraints = new GridBagConstraints();
        gridConstraints.gridx = 2;
        gridConstraints.gridy = 3;
        gridConstraints.gridwidth = 2;
        gridConstraints.anchor =
 GridBagConstraints.NORTH;
        getContentPane().add(factorPanel,
 gridConstraints); int x = 2;
        int y = 0;
        for (int i = 0; i < 11; i++)
{

       factorRadioButton[i] = new JRadioButton();
       factorRadioButton[i].setText(String.valueOf(i));
       factorRadioButton[i].setBackground(lightBlue);
       factorButtonGroup.add(factorRadioButton[i]);
       gridConstraints = new GridBagConstraints();
       if (i < 10) {
       gridConstraints.gridx = x;
```

```
    gridConstraints.gridy = y;
    }
    else {
    gridConstraints.gridx = 0;
    gridConstraints.gridy = 0;
    gridConstraints.gridwidth = 2;
    }
    gridConstraints.anchor = GridBagConstraints.WEST;
    factorPanel.add(factorRadioButton[i], 
gridConstraints);
    factorRadioButton[i].addActionListener(new 
ActionListener() {
    public void actionPerformed(ActionEvent e) {
    factorRadioButtonActionPerformed(e); }
    });
    x++;
    if(x > 2)
    {
    x = 0;
    y++;

    }
    }

     factorRadioButton[10].setText("Random");
     factorRadioButton[10].setSelected(true);
     timerPanel.setPreferredSize(new Dimension(130, 
130));
     timerPanel.setBorder
(BorderFactory.createTitledBorder("Timer:"));
     timerPanel.setBackground(lightBlue);
     timerPanel.setLayout(new GridBagLayout());
     gridConstraints = new GridBagConstraints();
     gridConstraints.gridx = 4;
     gridConstraints.gridy = 3;
     gridConstraints.insets = new Insets(0, 0, 0, 
10);
     gridConstraints.anchor = 
GridBagConstraints.NORTH;
     getContentPane().add(timerPanel, 
gridConstraints);
     for (int i = 0; i < 3; i++)
{
         timerRadioButton[i] = new JRadioButton();
         timerRadioButton[i].setBackground
(lightBlue);
         timerButtonGroup.add(timerRadioButton[i]);
         gridConstraints = new GridBagConstraints();
         gridConstraints.gridx = 0;
         gridConstraints.gridy = i;
         gridConstraints.gridwidth = 2;
         gridConstraints.anchor = GridBagConstraints.
WEST;
         timerPanel.add
(timerRadioButton[i],gridConstraints);
         timerRadioButton[i].addActionListener(new 
ActionListener() {

             public void actionPerformed(ActionEvent 
e) {
                 timerRadioButtonActionPerformed(e);
}
             });
}

     timerRadioButton[0].setText("Off");
     timerRadioButton[1].setText("On-Count Up");
     timerRadioButton[2].setText("On-CountDown");
     timerRadioButton[0].setSelected(true);
     timerTextField.setText("Off");
     timerTextField.setPreferredSize(new Dimension
(90,25));
     timerTextField.setEditable(false);
     timerTextField.setBackground(Color.WHITE);
     timerTextField.setForeground(Color.RED);
     timerTextField.setHorizontalAlignment
(SwingConstants.CENTER);
     timerTextField.setFont(myFont);
     gridConstraints = new GridBagConstraints();
     gridConstraints.gridx = 0;
     gridConstraints.gridy = 3;
     gridConstraints.anchor = GridBagConstraints.
WEST;
     gridConstraints.insets = new Insets(5, 0, 0, 0);
     timerPanel.add(timerTextField,gridConstraints);
     timerScrollBar.setPreferredSize(new Dimension
(20, 25));
     timerScrollBar.setMinimum(1);
     timerScrollBar.setMaximum(60);
     timerScrollBar.setValue(1);
     timerScrollBar.setBlockIncrement(1);
     timerScrollBar.setUnitIncrement(1);
     timerScrollBar.setOrientation
(JScrollBar.VERTICAL);
     timerScrollBar.setEnabled(false);
     gridConstraints = new GridBagConstraints();
     gridConstraints.gridx = 1;
     gridConstraints.gridy = 3;
     gridConstraints.anchor =
GridBagConstraints.WEST;
     gridConstraints.insets = new Insets(5, 0, 0, 0);
     timerPanel.add(timerScrollBar, gridConstraints);
     timerScrollBar.addAdjustmentListener(new
AdjustmentListener() {

```

```java
         public void adjustmentValueChanged(AdjustmentEvent e) {
             timerScrollBarAdjustmentValueChanged(e); }
           });
     startButton.setText("Start Practice");
     gridConstraints = new GridBagConstraints();
     gridConstraints.gridx = 0;
     gridConstraints.gridy = 4;
     gridConstraints.gridwidth = 2;
     gridConstraints.insets = new Insets(10, 0, 10, 0);
     getContentPane().add(startButton, gridConstraints);
     startButton.addActionListener(new ActionListener() {

         public void actionPerformed(ActionEvent e) {
             startButtonActionPerformed(e);
             }
         });
     exitButton.setText("Exit");
     gridConstraints = new GridBagConstraints();
     gridConstraints.gridx = 2;
     gridConstraints.gridy = 4;
     gridConstraints.gridwidth = 2;
     gridConstraints.insets = new Insets(10, 0, 10, 0);
     getContentPane().add(exitButton, gridConstraints);
     exitButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
exitButtonActionPerformed(e);
}
```

```
});
problemsTimer = new Timer (1000, new ActionListener()
{
public void actionPerformed(ActionEvent e) {
problemsTimerActionPerformed(e);
}
});
pack();
Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
setBounds((int)((screenSize.width - getWidth())),
 (int) ((screenSize.height - getHeight())), getWidth
 (), getHeight());}
private void exitForm(WindowEvent evt) {
System.exit(0);
}
private void typeCheckBoxActionPerformed(ActionEvent
e) {
int numberChecks;
int clickedBox = 0;
// determine which box was clicked
String s = e.getActionCommand();
if (s.equals("Addition"))
clickedBox = 0;
else if (s.equals("Subtraction")) clickedBox = 1;
else if (s.equals("Multiplication")) clickedBox = 2;
else if (s.equals("Division"))
clickedBox = 3;
// determine how many boxes are checked
numberChecks = 0;
if (typeCheckBox[0].isSelected()) numberChecks++;
if (typeCheckBox[1].isSelected()) numberChecks++;
if (typeCheckBox[2].isSelected()) numberChecks++;
if (typeCheckBox[3].isSelected()) {
    numberChecks++;
```

```
// make sure zero not selected factor
    if(factorRadioButton[0].isSelected())
        factorRadioButton[1].doClick();
        factorRadioButton[0].setEnabled(false); }
else
{
factorRadioButton[0].setEnabled(true); }
// if all boxes unchecked, recheck last clicked box
if (numberChecks == 0)
    typeCheckBox[clickedBox].setSelected(true);
    problemLabel.requestFocus();
    }
    private void factorRadioButtonActionPerformed
(ActionEvent e){
    problemLabel.requestFocus();
    }
    private void timerRadioButtonActionPerformed
(ActionEvent e) {
    if (timerRadioButton[0].isSelected()) {
    timerTextField.setText("Off");
    timerScrollBar.setEnabled(false); }
    else if (timerRadioButton[1].isSelected()) {
    problemTime = 0;
    timerTextField.setText(getTime(problemTime));
    timerScrollBar.setEnabled(false); }
    else if (timerRadioButton[2].isSelected()) {
    problemTime = 30 * timerScrollBar.getValue();
    timerTextField.setText(getTime(problemTime));
    timerScrollBar.setEnabled(true);
    }
    }
    private void timerScrollBarAdjustmentValueChanged
(AdjustmentEvent e) {
    timerTextField.setText(getTime(30 *
timerScrollBar.getValue()));}
```

```java

    private void startButtonActionPerformed(ActionEvent e){
    int score;
    String message = "";
    if (startButton.getText().equals("Start Practice")) {
        startButton.setText("Stop Practice");
        exitButton.setEnabled(false);
        numberTried = 0;
        numberCorrect = 0;
        triedTextField.setText("0");
        correctTextField.setText("0");
        timerRadioButton[0].setEnabled(false);
        timerRadioButton[1].setEnabled(false);
        timerRadioButton[2].setEnabled(false);
        timerScrollBar.setEnabled(false);
        if(!timerRadioButton[01].isSelected()) {
        if (timerRadioButton[1].isSelected()) problemTime = 0;
        else
        problemTime = 30 * timerScrollBar.getValue();
        timerTextField.setText(getTime(problemTime));
        problemsTimer.start();
        }
        problemLabel.setText(getProblem());}
    else
        {
        timerRadioButton[0].setEnabled(true);
        timerRadioButton[1].setEnabled(true);
        timerRadioButton[2].setEnabled(true);
        if (timerRadioButton[2].isSelected())
            timerScrollBar.setEnabled(true);
        problemsTimer.stop();
        startButton.setText("Start Practice");
        exitButton.setEnabled(true);
        problemLabel.setText("");
        if (numberTried > 0){
            score = (int) (100 * (double) (numberCorrect) / numberTried);
            message = "Problems Tried: " + String.valueOf(numberTried) + "\n";
            message += "Problems Correct: " + String.valueOf(numberCorrect) + " (" + String.valueOf(score) + " %)" + "\n";
            if(timerRadioButton[0].isSelected()) {
            message += "Timer Off";
            }
            else
            {
            if(timerRadioButton[2].isSelected()) {
            problemTime = 30 * timerScrollBar.getValue() - problemTime; }
            message += "Elapsed Time: " + getTime(problemTime) + "\n";
            message += "Time Per Problem: " + new DecimalFormat("0.00").format((double)(problemTime)/numberTried) + " sec";
            }
            JOptionPane.showConfirmDialog(null, message, "Results",
            JOptionPane.DEFAULT_OPTION, JOptionPane.INFORMATION_MESSAGE); }
        }
    }

            private void exitButtonActionPerformed(ActionEvent e){
            System.exit(0);
            }
            private void problemLabelKeyPressed(KeyEvent e) {
            if (startButton.getText().equals("Start Practice")) return;
            // only allow number keys
            if (e.getKeyChar() >= '0' && e.getKeyChar() <= '9') {
                yourAnswer += e.getKeyChar();
                problemLabel.setText(problem + yourAnswer);
                if (digitNumber != numberDigits) {
                    digitNumber++;
            problemLabel.setText(problemLabel.getText() + "?"); return;
                }
    else
    {
    numberTried++;
    // check answer
    if (Integer.valueOf(yourAnswer).intValue()==correctAnswer)
    {
    numberCorrect++;
    }
    triedTextField.setText(String.valueOf(numberTried));
    correctTextField.setText(String.valueOf(numberCorrect));
    problemLabel.setText(getProblem());}
        }
        }
    private void problemsTimerActionPerformed(ActionEvent e){
    if (timerRadioButton[1].isSelected()) {
        problemTime++;
    timerTextField.setText(getTime(problemTime));
    if (problemTime >= 1800) {
        startButton.doClick();
        return;
    }
    }
    else
    {
    problemTime--;
    timerTextField.setText(getTime(problemTime));
    if (problemTime == 0)
    {
    startButton.doClick();
    return;
    }
    }
    }

    private String getProblem() {


    int pType, p, number, factor;
    p = 0;
    do
        {
        pType = myRandom.nextInt(4) + 1;
    if (pType == 1 && typeCheckBox[0].isSelected()) {
    // Addition
        p = pType;
        number = myRandom.nextInt(10);
        factor = getFactor(1);
        correctAnswer = number + factor;
        problem = String.valueOf(number) + " + " +
String.valueOf(factor) + " = "; }
        else if (pType == 2 &&
typeCheckBox[1].isSelected()) {
        // Subtraction
        p = pType;
        factor = getFactor(2);
        correctAnswer = myRandom.nextInt(10);
        number = correctAnswer + factor;
        problem = String.valueOf(number) + " - " +
String.valueOf(factor) + " ="; }
        else if (pType == 3 &&
typeCheckBox[2].isSelected()) {
        // Multiplication
        p = pType;
        number = myRandom.nextInt(10);
        factor = getFactor(3);
        correctAnswer = number * factor;
        problem = String.valueOf(number) + " × " +
String.valueOf(factor) + " = "; }
        else if (pType == 4 &&
typeCheckBox[3].isSelected()) {
        // Division
        p = pType;
        factor = getFactor(4);
        correctAnswer = myRandom.nextInt(10);
        number = correctAnswer * factor;
        problem = String.valueOf(number) + " / " +
String.valueOf(factor) + " ="; }
        }
        while (p == 0);
            yourAnswer ="";
            digitNumber = 1;
            problemLabel.requestFocus();
            if (correctAnswer < 10)
            {
            numberDigits = 1;
            return (problem + "?");
```

```
                }
                else
                {
                numberDigits = 2;
                return (problem +"??");
                }
        }
                private int getFactor(int p)
                {
                if (factorRadioButton[10].isSelected()) {
                //random
                if (p == 4)
                return (myRandom.nextInt(9) + 1); else
                    return (myRandom.nextInt(10));
                }
                else
                {
                    for (int i = 0; i < 10; i++)
                    {
                    if (factorRadioButton[i].isSelected
()) 
                        return(i);
                    }
                    return (0);
                }
                }
                private String getTime(int s)
                {
                    int min, sec;
                    String ms, ss;
                    min = (int) (s / 60);
                    sec = s - 60 * min;
                    ms = String.valueOf(min);
                    ss = String.valueOf(sec);
                    if (sec < 10)
                    ss = "0" + ss;
                    return (ms + ":" + ss);

    }
    }
```

```
package MCE;

import javax.swing.filechooser.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.Random;
import java.text.*;
import java.awt.EventQueue;

import javax.swing.border.EmptyBorder;

public class MultipleChoiceExam extends JFrame {

    JLabel headGivenLabel = new JLabel();
    JLabel givenLabel = new JLabel();
    JLabel headAnswerLabel = new JLabel();
    JLabel[] answerLabel = new JLabel[4];
    JTextField answerTextField = new JTextField();
    JTextArea commentTextArea = new JTextArea();
    JButton nextButton = new JButton();
    JButton startButton = new JButton();
    // menu structure
    JMenuBar mainMenuBar = new JMenuBar();
    JMenu fileMenu = new JMenu("File");
    JMenuItem openMenuItem = new JMenuItem("Open");
    JMenuItem exitMenuItem = new JMenuItem("Exit");
    JMenu optionsMenu = new JMenu("Options");
    JRadioButtonMenuItem header1MenuItem = new
    JRadioButtonMenuItem("Header 1", true);
JRadioButtonMenuItem
    header2MenuItem = new JRadioButtonMenuItem
("Header 2", false);
    JRadioButtonMenuItem mcMenuItem = new
    JRadioButtonMenuItem("Multiple Choice Answers", true);
    JRadioButtonMenuItem typeMenuItem = new
    JRadioButtonMenuItem("Type In Answers", false);
    ButtonGroup nameGroup = new ButtonGroup();
    ButtonGroup tipoGroup = new ButtonGroup();
    Font headerFont = new Font("Arial", Font.BOLD, 18);
    Font
    examItemFont = new Font("Arial", Font.BOLD, 16);
    Dimension itemSize = new Dimension(370, 30);

    String examTitle;
    String header1, header2;
    int numberTerms;
    String[] term1 = new String[100];
    String[] term2 = new String[100];
    int numberTried, numberCorrect;
    int correctAnswer;
    Random myRandom = new Random();


    public static void main(String[] args) {
        new MultipleChoiceExam().~~show~~();
    }

    public MultipleChoiceExam() {
        setTitle("Multiple Choice Exam - No File");
        setResizable(false);
        addWindowListener(new WindowAdapter()
        {
        public void windowClosing(WindowEvent evt)
        {
        exitForm(evt);
        }
```

```java

        private void exitForm(WindowEvent evt) {
            // TODO Auto-generated method stub

        }
        });
        getContentPane().setLayout(new GridBagLayout());
        GridBagConstraints gridConstraints;
        headGivenLabel.setPreferredSize(itemSize);
        headGivenLabel.setFont(headerFont);
        gridConstraints = new GridBagConstraints();
        gridConstraints.gridx = 0;
        gridConstraints.gridy = 0;
        gridConstraints.insets = new Insets(10, 10, 0, 10);
        getContentPane().add(headGivenLabel, gridConstraints);
        givenLabel.setPreferredSize(itemSize);
        givenLabel.setFont(examItemFont);
        givenLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        givenLabel.setBackground(Color.WHITE);
        givenLabel.setForeground(Color.BLUE);
        givenLabel.setOpaque(true);
        givenLabel.setHorizontalAlignment(SwingConstants.CENTER);
        gridConstraints = new GridBagConstraints();
        gridConstraints.gridx = 0;
        gridConstraints.gridy = 1;
        gridConstraints.insets = new Insets(0, 10, 0, 10);
        getContentPane().add(givenLabel, gridConstraints);
        headAnswerLabel.setPreferredSize(itemSize);
        headAnswerLabel.setFont(headerFont);
        gridConstraints = new GridBagConstraints();
        gridConstraints.gridx = 0;
        gridConstraints.gridy = 2;
        gridConstraints.insets = new Insets(10, 10, 0, 10);
        getContentPane().add(headAnswerLabel, gridConstraints);
        for (int i= 0; i < 4; i++){
                answerLabel[i] = new JLabel();
                answerLabel[i].setPreferredSize(itemSize);
                answerLabel[i].setFont(examItemFont);
                answerLabel[i].setBorder(BorderFactory.createLineBorder(Color.BLACK));
                answerLabel[i].setBackground(Color.WHITE);
                answerLabel[i].setForeground(Color.BLUE);
                answerLabel[i].setOpaque(true);
                answerLabel[i].setHorizontalAlignment(SwingConstants.CENTER);
                gridConstraints = new GridBagConstraints();
                gridConstraints.gridx = 0;
                gridConstraints.gridy = i + 3;
                gridConstraints.insets = new Insets(0, 10, 10, 10);
                getContentPane().add(answerLabel[i], gridConstraints);
                answerLabel[i].addMouseListener(new MouseAdapter() {
                public void mousePressed(MouseEvent e)
                {
```

```
                mousePressed(e);
                }
                });
                }
                answerTextField.setPreferredSize
   (itemSize);
                answerTextField.setFont
   (examItemFont);
                answerTextField.setBackground
   (Color.WHITE);
                answerTextField.setForeground
   (Color.BLUE);
                answerTextField.setVisible(false);
                gridConstraints = new
   GridBagConstraints();
                gridConstraints.gridx = 0;
                gridConstraints.gridy = 3;
                gridConstraints.insets = new Insets
   (0, 10, 10, 10);
                getContentPane().add(answerTextField,
   gridConstraints);
                answerTextField.addActionListener(new
   ActionListener () {
                public void actionPerformed
   (ActionEvent e)
                {
                actionPerformed(e);
                }
                });
                commentTextArea.setPreferredSize(new
   Dimension(370, 80));
                commentTextArea.setFont(new Font
   ("Courier New", Font.BOLD +
                Font.ITALIC, 18));
                commentTextArea.setBorder
```

```
(BorderFactory.createLineBorder(Color.BLACK));
                commentTextArea.setEditable(false);
                commentTextArea.setBackground(new
Color(255, 255, 192));
                commentTextArea.setForeground
(Color.RED);
                gridConstraints = new
GridBagConstraints();
                gridConstraints.gridx = 0;
                gridConstraints.gridy = 7;
                gridConstraints.insets = new Insets
(0, 10, 10, 10);
                getContentPane().add(commentTextArea,
gridConstraints);
                nextButton.setText("Next Question");
                gridConstraints = new
GridBagConstraints();
                gridConstraints.gridx = 0;
                gridConstraints.gridy = 8;
                gridConstraints.insets = new Insets
(0, 0, 10, 0);
                getContentPane().add(nextButton,
gridConstraints);
                nextButton.addActionListener(new
ActionListener() {
                public void actionPerformed
(ActionEvent e)
                {
                actionPerformed(e);
                }
                });
                startButton.setText("Start Exam");
                gridConstraints = new
GridBagConstraints();
                gridConstraints.gridx = 0;
```

```
        gridConstraints.gridy = 9;
        gridConstraints.insets = new Insets(0, 0, 10, 0);
        getContentPane().add(startButton, gridConstraints);
        startButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
        actionPerformed(e);
        }
        });
        // build menu structure
        setJMenuBar(mainMenuBar);
        mainMenuBar.add(fileMenu);
        fileMenu.add(openMenuItem);
        fileMenu.addSeparator();
        fileMenu.add(exitMenuItem);
        mainMenuBar.add(optionsMenu);
        optionsMenu.add(header1MenuItem);
        optionsMenu.add(header2MenuItem);
        optionsMenu.addSeparator();
        optionsMenu.add(mcMenuItem);
        optionsMenu.add(typeMenuItem);
        nameGroup.add(header1MenuItem);
        nameGroup.add(header2MenuItem);
        tipoGroup.add(mcMenuItem);
        tipoGroup.add(typeMenuItem);
        openMenuItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
            actionPerformed(e);
```

```
}
});
exitMenuItem.addActionListener(new
 ActionListener() {
public void actionPerformed
 (ActionEvent e)
{
actionPerformed(e);
}
});
header1MenuItem.addActionListener(new
 ActionListener() {
public void actionPerformed
 (ActionEvent e)
{
actionPerformed(e);
}
});
header2MenuItem.addActionListener(new
 ActionListener() {
public void actionPerformed
 (ActionEvent e)
{
actionPerformed(e);
}
});
mcMenuItem.addActionListener(new
 ActionListener() {
public void actionPerformed
 (ActionEvent e)
{
actionPerformed(e);
}
});
typeMenuItem.addActionListener(new
ActionListener() {
                    public void actionPerformed
(ActionEvent e)
                    {
                    actionPerformed(e);
                    }
                    });
                    pack();
                    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
                    setBounds((int) (0.5*
(screenSize.width - getWidth())), (int) (0.5*
(screenSize.height - getHeight())), getWidth(),
getHeight()); // initialize form
                    startButton.setEnabled(false);
                    nextButton.setEnabled(false);
                    optionsMenu.setEnabled(false);
                    commentTextArea.setText
(centerTextArea("Open Exam File toStart")); }
                    private void exitForm(WindowEvent
evt)
                    {
                    System.exit(0);
                    }
                    private void answerLabelMousePressed
(MouseEvent e) {
                    boolean correct = false;
                    int labelSelected;
                    // make sure exam has started and
question has not been answered if
                    if(startButton.getText().equals
("Start Exam") || nextButton.isEnabled())
                    return;
                    // determine which label was clicked
                    // get upper left corner of clicked
```

```
label
                    Point p = e.getComponent
().getLocation();
                    // determine index based on p
                    for (labelSelected = 0; labelSelected
< 20; labelSelected++) {
                    if (p.x ==
answerLabel[labelSelected].getX() && p.y ==
                    answerLabel[labelSelected].getY())
break;
                    }
                    // If already answered, exit
                    numberTried++;
                    if (header1MenuItem.isSelected())
                    {
                    if
(answerLabel[labelSelected].getText().equals
(term1[correctAnswer])) correct = true;
                    }
                    else
                    {
                    if
(answerLabel[labelSelected].getText().equals
(term2[correctAnswer])) correct = true;
                    }
                    updateScore(correct);
                    }
                    private void
answerTextFieldActionPerformed(ActionEvent e) {
                    // Check type in answer
                    boolean correct;
                    String ucTypedAnswer, ucAnswer;
                    // make sure exam has started and
question has not been answered if
                    if(startButton.getText().equals
("Start Exam") || nextButton.isEnabled())
                    return;
                    answerTextField.setEditable(false);
                    numberTried++;
                    ucTypedAnswer =
answerTextField.getText().toUpperCase(); if
                    (header1MenuItem.isSelected())
                    ucAnswer =
term1[correctAnswer].toUpperCase();
                    else
                    ucAnswer =
term2[correctAnswer].toUpperCase();
                    correct = false;
                    if (ucTypedAnswer.equals(ucAnswer) ||
                    soundex(ucTypedAnswer).equals(soundex
(ucAnswer))) correct = true;
                    updateScore(correct);
                    }
                    private void
nextButtonActionPerformed(ActionEvent e) {
                    // Generate next question
                    nextButton.setEnabled(false);
                    nextQuestion();
                    }
                    private void
startButtonActionPerformed(ActionEvent e) {
                    String message;
                    if (startButton.getText().equals
("Start Exam"))
                    {
                    startButton.setText("Stop Exam");
                    nextButton.setEnabled(false);
                    // Reset the score
                    numberTried = 0;
                    numberCorrect = 0;
```

```
                        commentTextArea.setText("");
                        fileMenu.setEnabled(false);
                        optionsMenu.setEnabled(false);
                        nextQuestion();
                        }
                        else
                        {
                        startButton.setText("Start Exam");
                        nextButton.setEnabled(false);
                        if (numberTried > 0)
                        {
                        message = "Questions Tried: " +
    String.valueOf(numberTried)
                        + "\n"; message += "Questions
    Correct: " +
                        String.valueOf(numberCorrect) + "\n
    \n"; message += "Your Score: " +
                        new DecimalFormat("0.0").format(100.0
    * ((double) numberCorrect /
                        numberTried)) + "%";
    JOptionPane.showConfirmDialog(null, message,
                        examTitle + " Results",
    JOptionPane.DEFAULT_OPTION,
                        JOptionPane.INFORMATION_MESSAGE); }
                        givenLabel.setText("");
                        answerLabel[0].setText("");
                        answerLabel[1].setText("");
                        answerLabel[2].setText("");
                        answerLabel[3].setText("");
                        answerTextField.setText("");
                        commentTextArea.setText
    (centerTextArea("ChooseOptions\nClick Start Exam"));
    fileMenu.setEnabled(true);
                        optionsMenu.setEnabled(true);
                        }
```

```
            }
            private void
openMenuItemActionPerformed(ActionEvent e) {
            String myLine;
            JFileChooser openChooser = new
JFileChooser();
            openChooser.setDialogType
(JFileChooser.OPEN_DIALOG);
            openChooser.setDialogTitle("Open Exam
File");
            openChooser.addChoosableFileFilter
(new
            FileNameExtensionFilter("Exam Files",
"csv")); if
            (openChooser.showOpenDialog(this) ==
            JFileChooser.APPROVE_OPTION) {
            try
            {
            BufferedReader inputFile = new
BufferedReader(new
            FileReader
(openChooser.getSelectedFile())); myLine =
            inputFile.readLine();
            examTitle = parseLeft(myLine);
            myLine = inputFile.readLine();
            header1 = parseLeft(myLine);
            header2 = parseRight(myLine);
            numberTerms = 0;
            do
            {
            numberTerms++;
            myLine = inputFile.readLine();
            term1[numberTerms - 1] = parseLeft
(myLine);
            term2[numberTerms - 1] = parseRight
```

```
(myLine);
                    }
                    while (inputFile.ready() &&
numberTerms < 100); if
                    (numberTerms < 5)
                    {
                    JOptionPane.showConfirmDialog(null,
"Must have at least 5 entries in exam file.", "Exam
File Error",
                    JOptionPane.DEFAULT_OPTION,
JOptionPane.ERROR_MESSAGE);
                    return;
                    }
                    inputFile.close();
                    // establish frame title
                    this.setTitle("Multiple Choice Exam -
" + examTitle); // set up menu items
                    header1MenuItem.setText(header1 + ",
Given " + header2);
                    header2MenuItem.setText(header2 + ",
Given " + header1); if
                    (header1MenuItem.isSelected())
                    {
                    headGivenLabel.setText(header2);
                    headAnswerLabel.setText(header1);
                    }
                    else
                    {
                    headGivenLabel.setText(header1);
                    headAnswerLabel.setText(header2);
                    }
                    startButton.setEnabled(true);
                    optionsMenu.setEnabled(true);
                    commentTextArea.setText
(centerTextArea("File Loaded,Choose Options\nClick
Start Exam")); }
                    catch (Exception ex)
                    {
                    JOptionPane.showConfirmDialog(null,
"Error reading in input file - make sure file is
correct format.", "Multiple Choice Exam File Error",
JOptionPane.DEFAULT_OPTION,
                    JOptionPane.ERROR_MESSAGE); return;
                    }
                    }
                    }
                    private void
exitMenuItemActionPerformed(ActionEvent e) {
                    System.exit(0);
                    }
                    private void
header1MenuItemActionPerformed(ActionEvent e) {
                    // Set up for naming header1, given
header2
                    headGivenLabel.setText(header2);
                    headAnswerLabel.setText(header1);
                    }
                    private void
header2MenuItemActionPerformed(ActionEvent e) {
                    // Set up for naming header2, given
header1
                    headGivenLabel.setText(header1);
                    headAnswerLabel.setText(header2);
                    }
                    private void
mcMenuItemActionPerformed(ActionEvent e) {
                    answerLabel[0].setVisible(true);
                    answerLabel[1].setVisible(true);
                    answerLabel[2].setVisible(true);
                    answerLabel[3].setVisible(true);
```

```
        answerTextField.setVisible(false);
        }
        private void typeMenuItemActionPerformed(ActionEvent e) {
        answerLabel[0].setVisible(false);
        answerLabel[1].setVisible(false);
        answerLabel[2].setVisible(false);
        answerLabel[3].setVisible(false);
        answerTextField.setVisible(true);
        }
        private String parseLeft(String s)
        {
        int cl;
        // find comma
        cl = s.indexOf(",");
        return (s.substring(0, cl));
        }
        private String parseRight(String s)
        {
        int cl;
        // find comma
        cl = s.indexOf(",");
        return (s.substring(cl + 1));
        }
        private String centerTextArea(String s)
        {
        // centers up to two lines in text area
        int charsPerLine = 33;
        String sOut = "";
        int j = s.indexOf("\n");
        int nSpaces;
        if (j == -1)
        {
```

```
// single line
sOut = "\n" + spacePadding((int)
((charsPerLine - s.length()) / 2))
+ s; }
else
{
// first line
String l = s.substring(0, j);
sOut = "\n" + spacePadding((int)
((charsPerLine - l.length()) / 2))
+ l; // second line
l = s.substring(j + 1);
sOut += "\n" + spacePadding((int)
((charsPerLine - l.length()) / 2))
+ l ; }
return(sOut);
}
private String spacePadding(int n)
{
String s = "";
if (n != 0)
for (int i = 0; i < n; i++)
s += " ";
return(s);
}
private void nextQuestion()
{
boolean[] termUsed = new
boolean[numberTerms];
int[] index = new int[4];
int j;
commentTextArea.setText("");
// Generate the next question based
on selected options
correctAnswer = myRandom.nextInt
```

```
(numberTerms);
                    if (header1MenuItem.isSelected())
                    {
                    givenLabel.setText
(term2[correctAnswer]);
                    }
                    else
                    {
                    givenLabel.setText
(term1[correctAnswer]);
                    }
                    if (mcMenuItem.isSelected())
                    {
                    // Multiple choice answers
                    for (int i = 0; i < numberTerms; i++)
                    {
                    termUsed[i] = false;
                    }

                    // Pick four random possiblities
                    for (int i = 0; i < 4; i++)
                    {
                    do
                    {
                    j = myRandom.nextInt(numberTerms);
                    }
                    while (termUsed[j] || j ==
correctAnswer);
                    termUsed[j] = true;
                    index[i] = j;
                    }
                    // Replace one with correct answer
                    index[myRandom.nextInt(4)] =
correctAnswer;
                    // Display multiple choice answers in
label boxes if
                    if(header1MenuItem.isSelected())
                    {
                    answerLabel[0].setText
(term1[index[0]]);
                    answerLabel[1].setText
(term1[index[1]]);
                    answerLabel[2].setText
(term1[index[2]]);
                    answerLabel[3].setText
(term1[index[3]]);
                    }
                    else
                    {
                    answerLabel[0].setText
(term2[index[0]]);
                    answerLabel[1].setText
(term2[index[1]]);
                    answerLabel[2].setText
(term2[index[2]]);
                    answerLabel[3].setText
(term2[index[3]]);
                    }
                    }
                    else
                    {
                    // Type-in answers
                    answerTextField.setEditable(true);
                    answerTextField.setText("");
                    answerTextField.requestFocus();
                    }
                    }
                    private void updateScore(boolean
correct)
                    {
```

```
                // Check if answer is correct
                if (correct)
                {
                numberCorrect++;
                commentTextArea.setText
(centerTextArea("Correct!")); }
                else
                commentTextArea.setText
(centerTextArea("Sorry ... CorrectAnswer Shown")); //
Display correct answer
                if (mcMenuItem.isSelected())
                {
                if (header1MenuItem.isSelected())
                answerLabel[0].setText
(term1[correctAnswer]);
                else
                answerLabel[0].setText
(term2[correctAnswer]);
                answerLabel[1].setText("");
                answerLabel[2].setText("");
                answerLabel[3].setText("");
                }
                else
                {
                if (header1MenuItem.isSelected())
                answerTextField.setText
(term1[correctAnswer]);
                else
                answerTextField.setText
(term2[correctAnswer]);
                }
                startButton.setEnabled(true);
                nextButton.setEnabled(true);
                nextButton.requestFocus();
                }
```

```
            public String soundex(String w)
            {
            // Generates Soundex code for W based
on Unicode value // Allowsanswers whose spelling is
close, but not exact
                String wTemp, s = "";
            int l;
            int wPrev, wSnd, cIndex;
            // Load soundex function array
            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,
            2, 0, 2}; wTemp = w.toUpperCase();
            l = w.length();
            if (l != 0)
            {
            s = String.valueOf(w.charAt(0));
            wPrev = 0;
            if (l > 1)
            {
            for (int i = 1; i < l; i++)
            {
            cIndex = (int) wTemp.charAt(i) - 65;
            if (cIndex >= 0 && cIndex <= 25)
            {
            wSnd = wSound[cIndex] + 48;
            if (wSnd != 48 && wSnd != wPrev)
            {
            s += String.valueOf((char) wSnd);
            }
            wPrev = wSnd;
            }
            }
            }
            else s=" ";{
                return(s);
                }
            }
            return s;

        }
}

```