```java
package firstGraph;

import java.awt.Color;
import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.Timer;

@SuppressWarnings("serial")
public class MainWindow extends JFrame implements ActionListener,
        MouseListener, MouseMotionListener {

    public int currentSlide = 1; // Number of slide that is loaded at launch

    AA_FirstSlide aaInstance; // Each slide has its own class.
    AB_SecondSlide abInstance; // Each class is placed in separate file.
    AC_ThirdSlide acInstance;// All the classes are sub-classes of
    AD_FourthSlide adInstance;// Variables class.
    AE1_FifthSlide ae1Instance;// These classes are declared here.
    AE2_FifthSlide ae2Instance;
    AF1_SixthSlide af1Instance;
    AF2_SixthSlide af2Instance;
    AG_SeventhSlide agInstance;
    AH_EightSlide ahInstance;
    AI_NinthSlide aiInstance;
    AJ_TenthSlide ajInstance;
    AK_EleventhSlide akInstance;
    AL_TwelfthSlide alInstance;
    AM_ThirteenthSlide amInstance;
    AN_FourteenthSlide anInstance;
    AO_FifteenthSlide aoInstance;
    AP_SixteenthSlide apInstance;
    AQ_SeventeenthSlide aqInstance;
    AR_EighteenthSlide arInstance;
    AS_NinteenthSlide asInstance;
    AT_TwentiethSlide atInstance;
    AU_21stSlide auInstance;
    AV_22ndSlide avInstance;
    AW_23rdSlide awInstance;
    AX_24thSlide axInstance;
    AY_25thSlide ayInstance;
    AZ_26thSlide azInstance;
    BA1_27thSlide ba1Instance;
    BB1_28thSlide bb1Instance;
    BC1_29thSlide bc1Instance;
    BD1_30thSlide bd1Instance;
    BE1_31stSlide be1Instance;
    BF1_32ndSlide bf1Instance;
    BG1_33rdSlide bg1Instance;
    BA2_27thSlide ba2Instance;
    BB2_28thSlide bb2Instance;
    BC2_29thSlide bc2Instance;
    BD2_30thSlide bd2Instance;
    BE2_31stSlide be2Instance;
    BF2_32ndSlide bf2Instance;

    String line; // Used when displaying current line to the JTextArea
    int currentChar = 0; // Lastly displayed character.
    int currentLine = 0; // Line currently being displayed
    int totalLines; // Total lines in the text file.
    int answerTries; // Counts the answer tries. After 5 displays hint.

    Timer letterTimer; // Timer responsible for displaying letters one by one.
    int letterTimerDelay = 1000;

    Timer buttonDisplayDelayTimer; // Timer responsible for displaying buttons
    int buttonTimerDelay = 2000;// with delay.

    JPanel contentPane;// Content pane holding the buttons and the textArea.

    public MainWindow ()/**/{ // Constructor.
        setUpFrame();// Method used to display the window.
        initiateInstances();// This method must be run AFTER the setUpFrame
                            // method, because when constructing each class, the
                            // screen width and height are used.
        loadUpContent(getCurrentInstance());// Loads up the actuall content of
                                            // current slide.
    }

    private void initiateInstances () {// Creates instance of each slide-class
                                      // and both timers.
        aaInstance = new AA_FirstSlide();
        abInstance = new AB_SecondSlide();
        acInstance = new AC_ThirdSlide();
        adInstance = new AD_FourthSlide();
        ae1Instance = new AE1_FifthSlide();
        af1Instance = new AF1_SixthSlide();
        ae2Instance = new AE2_FifthSlide();
        af2Instance = new AF2_SixthSlide();
        agInstance = new AG_SeventhSlide();
        ahInstance = new AH_EightSlide();
        aiInstance = new AI_NinthSlide();
        ajInstance = new AJ_TenthSlide();
        akInstance = new AK_EleventhSlide();
        alInstance = new AL_TwelfthSlide();
        amInstance = new AM_ThirteenthSlide();
        anInstance = new AN_FourteenthSlide();
        aoInstance = new AO_FifteenthSlide();
        apInstance = new AP_SixteenthSlide();
        aqInstance = new AQ_SeventeenthSlide();
        arInstance = new AR_EighteenthSlide();
        asInstance = new AS_NinteenthSlide();
        atInstance = new AT_TwentiethSlide();
        auInstance = new AU_21stSlide();
        avInstance = new AV_22ndSlide();
        awInstance = new AW_23rdSlide();
        axInstance = new AX_24thSlide();
        ayInstance = new AY_25thSlide();
        azInstance = new AZ_26thSlide();
        ba1Instance = new BA1_27thSlide();
        bb1Instance = new BB1_28thSlide();
        bc1Instance = new BC1_29thSlide();
        bd1Instance = new BD1_30thSlide();
        be1Instance = new BE1_31stSlide();
        bf1Instance = new BF1_32ndSlide();
        bg1Instance = new BG1_33rdSlide();
        ba2Instance = new BA2_27thSlide();
        bb2Instance = new BB2_28thSlide();
        bc2Instance = new BC2_29thSlide();
        bd2Instance = new BD2_30thSlide();
        be2Instance = new BE2_31stSlide();
        bf2Instance = new BF2_32ndSlide();

        letterTimer = new Timer(getCurrentInstance().TIMER_SPEED,
                MainWindow.this);
        buttonDisplayDelayTimer = new Timer(200, MainWindow.this);
    }

    private Variables getCurrentInstance() {// Returns correct class instance
                                            // depending on slide currently
                                            // displayed.
                                            // Most used method through whole
                                            // program. Used in every method,
                                            // that is shared by multiple
                                            // slides.
        switch (currentSlide) {
        case 1:
            return aaInstance;
        case 2:
            return abInstance;
        case 3:
            return acInstance;
        case 4:
            return adInstance;
        case 5:
            return ae1Instance;
        case 6:
            return ae2Instance;
        case 7:
            return af1Instance;
        case 8:
            return af2Instance;
        case 9:
            return agInstance;
        case 10:
            return ahInstance;
        case 11:
            return aiInstance;
        case 12:
            return ajInstance;
        case 13:
            return akInstance;
        case 14:
            return alInstance;
        case 15:
            return amInstance;
        case 16:
            return anInstance;
        case 17:
            return aoInstance;
        case 18:
            return apInstance;
        case 19:
            return aqInstance;
        case 20:
            return arInstance;
        case 21:
            return asInstance;
        case 22:
            return atInstance;
        case 23:
            return auInstance;
        case 24:
            return avInstance;
        case 25:
            return awInstance;
        case 26:
            return axInstance;
        case 27:
            return ayInstance;
        case 28:
            return azInstance;
        case 29:
            return ba1Instance;
        case 30:
            return bb1Instance;
        case 31:
            return bc1Instance;
        case 32:
            return bd1Instance;
        case 33:
            return be1Instance;
        case 34:
            return bf1Instance;
        case 35:
            return bg1Instance;
        case 36:
            return ba2Instance;
        case 37:
            return bb2Instance;
        case 38:
            return bc2Instance;
        case 39:
            return bd2Instance;
        case 40:
            return be2Instance;
        case 41:
            return bf2Instance;

        default:
            System.out.println("Swear word, that was"
                    + " here used for testing purposes "
                    + "was replaced before hand-in.");
            return null;
        }
    }

    private int getAnwerSheet(Variables instance) {  // Returns answer sheet
                                                     // corresponding to the
                                                     // slide with question.
        if (instance == awInstance) {
            return 0;
        } else if (instance == bd1Instance) {
            return 1;
        } else if (instance == bd2Instance) {
            return 2;
        } else
            return -1;
    }

    private void setUpFrame() { // This method is called only once (when program
                                // is launched).

        setUndecorated(true); // No border around the window (to be fullscreem).
        setExtendedState(JFrame.MAXIMIZED_BOTH);// Fullscreen
        setSize(new Dimension(Toolkit.getDefaultToolkit().getScreenSize()));//
Fullscreen
        setResizable(false); // not resizable
        setVisible(true); // Visible
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);// Application is
                                                        // terminated after
                                                        // closing.
        setBackground(Color.BLACK); // background color
        Main.dimX = getWidth(); // Assigning screen width and height to the
                                // variable, which
        Main.dimY = getHeight(); // is used elsewhere in program.
        Variables.DEFAULT_BUTTON_WIDTH = Variables.calculatePos(9, true); // Sets
                                                                           // the
                                                                           //
default
                                                                           //
button
                                                                           // size
                                                                           //
depending
                                                                           // on
                                                                           //
screen
                                                                           // res.
        Variables.DEFAULT_BUTTON_HEIGHT = Variables.calculatePos(2, false);
        if (Main.dimX <= 1200 || Main.dimY <= 700) { // If screen is small,
                                                        // makes font smaller.
            Variables.BUTTON_FONT_SIZE = 20;
            Variables.TEXT_FONT_SIZE = 24;
        }
        addMouseMotionListener(this); // Motion listener used to change cursor
                                        // back to default (after changing to
                                        // hand cursor.
    }

    private void loadUpContent(Variables currentInstance) {// Loads up the
                                                            // background image.
                                                            // Then load
        // the content of content pane (buttons,
        // text...)
        loadImage(currentInstance); // Method used for loading the image
        setUpContentPane(currentInstance);// Method used for setting up the
                                            // content pane.
        // Method creating buttons is called from this
        // method.
        setUpTextArea(currentInstance);// Method used for setting up text
                                        // area (it there is one).
        setUpInputField(currentInstance); // Method used for creating input
                                            // field (if there is one).
        contentPane.add(currentInstance.label);
        refresh(); // Recalculating and repainting the frame.
        letterTimer.setInitialDelay(letterTimerDelay);// Setting the waiting
        buttonDisplayDelayTimer.setInitialDelay(buttonTimerDelay);// time,
                                                                    // before
                                                                    // timer
        // action is run.

    }

    private void loadImage(Variables currentInstance) {    // Method loading the
                                                        // background image
        try { // Following needs to be surrounded in try-catch in case the file
                // was corrupted (for example).
            currentInstance.image1 = ImageIO.read(new File(currentInstance
                    .getPIC_PATH())); // Reads pic to file
            currentInstance.myImageIcon = new ImageIcon( // "Transforms" file to
                                                            // displayable icon,
                    currentInstance.image1.getScaledInstance(-1, Main.dimY, //
Resizes
                                                                            // icon
                                                                            // to
                                                                            // fit
                                                                            // the
                                                                            //
screen.
                            Image.SCALE_SMOOTH));
        } catch (IOException e) {
            System.err.println("Error loading picture.");
            System.exit(0);
        } catch (NullPointerException e) {
            System.err.println("Error loading picture NPE.");
            System.exit(0);
        }

        currentInstance.label = new JLabel(); // Creates a label
        currentInstance.label.setIcon(currentInstance.myImageIcon); // Attaches
                                                                    // the icon
                                                                    // to this
                                                                    // label
        currentInstance.label.setBounds(
                // And places the label to fit the screen.
                (Main.dimX - currentInstance.myImageIcon.getIconWidth()) / 2,
                0, Main.dimX, Main.dimY);
    }

    private void setUpContentPane(Variables currentInstance) { // Sets up
                                                                // content pane
                                                                // (the "glass"
                                                                // of the
                                                                // window)
        contentPane = new JPanel(); // Create
        contentPane.setBackground(Color.BLACK); // Background
        contentPane.setBorder(null); // border
        setContentPane(contentPane);
        contentPane.setLayout(null); // sets no layout (pre-defined layouts are
                                        // unusable for us).

        for (int i = 0; i < currentInstance.getButtonCount(); i++) {
            setUpButton(
                    currentInstance,
                    i, // Loop that calls the method creating buttons.
                    currentInstance.getButtonSetUp(i).posX, // It uses the
                                                            // properties from
                                                            // currend slide
                                                            // class.
                    currentInstance.getButtonSetUp(i).posY, // It passes them to
                                                            // the setUpButton
                                                            // method as
```

```
                                                // parameters.
                    currentInstance.getButtonSetUp(i).width,
                    currentInstance.getButtonSetUp(i).heigth,
                    currentInstance.getButtonSetUp(i).caption,
                    currentInstance.getButtonSetUp(i).font,
                    currentInstance.getButtonSetUp(i).icon,
                    currentInstance.getButtonSetUp(i).visible);
        }
    }

    private void setUpButton(Variables currentInstance, int id, int positionX,
            int positionY, int width, int height, String name, Font font,
            String icon, boolean visible) {// Method for creating buttons with
        // button properties as parameters.
        currentInstance.button[id] = new JButton(name); // creates a button
        currentInstance.button[id].setBorder(null);// border
        currentInstance.button[id].setHorizontalAlignment(SwingConstants.LEFT);//
text
                                                                            //
inside
                                                                            //
the
                                                                            //
button
                                                                            //
alignment.
        currentInstance.button[id].setContentAreaFilled(false);// the button is
                                                        // transparent
        currentInstance.button[id].setOpaque(false);// transparency
        currentInstance.button[id].setForeground(Color.WHITE);// font color
        if (font != null) {// some buttons have only icon, hence no font
            currentInstance.button[id].setFont(font);
        }
        if (icon != null) {// some buttons dont have icon...
            currentInstance.button[id].setIcon(new ImageIcon(icon));
        }
        if (visible == false) { // some buttons are not visible from the
                                // beginning
            currentInstance.button[id].setVisible(false);
        }
        currentInstance.button[id].setBounds(positionX, positionY, width,
                height); // sets button position

        currentInstance.button[id].addMouseListener(this); // makes button
                                                      // click-sensitive
        currentInstance.button[id].addMouseMotionListener(this);// makes button
                                                        // motion-sensitive

        contentPane.add(currentInstance.button[id]);// finally, adds button to
                                                // the content pane
    }

    private void setUpTextArea(Variables currentInstance) {// sets up text area
        if (currentInstance.getTextSetUp() != null) { // if there is no text
                                                  // area defined in slide
                                                  // class, nothing
                                                  // happens.
            currentInstance.textArea = new JTextArea(); // creates text
                                                  // Area
            currentInstance.textArea.setBounds(
                    // positions the tet area
                    currentInstance.getTextSetUp().posX,
                    currentInstance.getTextSetUp().posY,
                    currentInstance.getTextSetUp().width,
                    currentInstance.getTextSetUp().heigth);

            currentInstance.textArea.setWrapStyleWord(true); // breaks the text
                                                                    // into multiple
                                                                    // lines
            currentInstance.textArea.setLineWrap(true);
            currentInstance.textArea.setEditable(false); // is not editable
            currentInstance.textArea.setText(currentInstance.text);// sets text

            if (currentInstance.getTextSetUp().someColor != null) {// if the
                                                                        // color is
                                                                        // specified,
                                                                        // uses the
                                                                        // color.
                currentInstance.textArea.setForeground(currentInstance
                        .getTextSetUp().someColor);
            } else
                // Otherwise uses default color.
                currentInstance.textArea
                        .setForeground(Variables.DEF_TEXT_COLOR);

            if (currentInstance == abInstance) { // Only second slide has Red
                                                    // text color.
                currentInstance.textArea.setBackground(new Color((float) 0.37,
                        (float) 0.37, (float) 0.37, (float) 0.7));
            } else { // all other slides have default text color
                currentInstance.textArea
                        .setBackground(Variables.DEF_TEXTAREA_COLOR);
            }
            currentInstance.textArea // sets the defined font.
                    .setFont(currentInstance.getTextSetUp().font);

            contentPane.add(currentInstance.textArea); // Adds text area to
                                                        // content pane.

            openFile(getCurrentInstance());// opens a file with text
        }
    }

    private void openFile(Variables currentInstance) { // opens file
        try { // Following needs to be surrounded in try-catch in case the file
                // was corrupted (for example).
            currentInstance.myFile = new File(currentInstance.getFILE_PATH());
            currentInstance.filRead = new FileReader(currentInstance.myFile);//
sets
                                                                                //
file
                                                                                //
reader
            currentInstance.bufRead = new BufferedReader(// buffered reader
                    currentInstance.filRead);
            totalLines = Integer.parseInt(currentInstance.bufRead.readLine());//
the
                                                                                //
first
                                                                                //
line
                                                                                //
in
                                                                                //
file
                                                                                //
has
                                                                                //
```

```
to
                                                                                  //
be
                                                                                  //
a
                                                                                  //
number
                                                                                  //
with
                                                                                  //
total
                                                                                  //
lines.
                                                                                  //
Reads
                                                                                  //
the
                                                                                  //
number
            currentLine = 0; // Resets the current line counter.
        } catch (IOException exception1) {
            System.err.println("Error in accesing text file.");
            System.out.println(exception1.getMessage());
            System.exit(1); // exits app
        } catch (NumberFormatException exception2) { // Error when the number of
                                                        // lines is not present.
            System.err.println("File lacking first line.");
            System.out.println(exception2.getMessage());
            System.exit(1); // exits app
        }
    }


    private void readLine(Variables currentInstance) {// reads line when called
        try {
            line = currentInstance.bufRead.readLine();// tries reading one line
                                                        // to string

        } catch (NullPointerException ex) {
            System.out.println("Error reading file");// not expected error
        } catch (IOException exception) {
            System.err.println("Error reading file.");
            System.out.println(exception.getMessage());
            System.exit(1);// exits
        }
        currentLine++; // Increments the current line counter.
        try {
            if (currentLine == totalLines) { // after all lines are displayed,
                                                // the button is displayed (with
                                                // delay).
                buttonDisplayDelayTimer.start();
            }
            if (currentLine == totalLines
                    && currentInstance.getButtonSetUp(currentInstance
                            .getButtonCount() - 1).icon
                            .equals(Variables.arrowPath)) {// If there is a
                                                            // button for
                                                            // displaying other
                                                            // part of text (in
                                                            // minority of
                                                            // slides, this
                                                            // hides it.
                currentInstance.button[currentInstance.getButtonCount() - 1]
                        .setVisible(false);
```

```java
            buttonDisplayDelayTimer.start();
        }
    } catch (NullPointerException exception) {
    }

    currentChar = 0; // resets line counter
    currentInstance.text = ""; // wipes the content of text area, so the
                                // animation of adding text letter-by-letter
                                // can be played
    letterTimer.start(); // starts the animation.
}

private void addLetter(Variables currentInstance) {// adds one letter -
                                                    // animation
    if (currentChar >= line.length()) {// keeps adding letters until all
                                        // line is displayed
        letterTimer.stop();
    } else {
        if (line.charAt(currentChar) == '¬') {// special char in text files
                                                // used to break text into
                                                // multiple lines
            currentInstance.text = currentInstance.text + "\n"; // adds
                                                                // breakpoint
                                                                // to
                                                                // string.
        } else {
            currentInstance.text = currentInstance.text
                    + line.charAt(currentChar); // adds letter to string
        }

        currentChar++;
        currentInstance.textArea.setText(currentInstance.text); // sets the
                                                                // changed
                                                                // string as
                                                                // text area
                                                                // text
        currentInstance.textArea.revalidate(); // Recalculates the text
                                                // area.
        refresh();
    }
}

private void setUpInputField(Variables currentInstance) {// sets up input
                                                            // field
    if (currentInstance.getImputSetUp() != null) {// if there is not input
                                                    // field specified,
                                                    // nothing happens
        currentInstance.inputField = new JTextField();// creates the input
                                                        // field
        currentInstance.inputField.setBounds(
                // positions it.
                currentInstance.getImputSetUp().posX,
                currentInstance.getImputSetUp().posY,
                currentInstance.getImputSetUp().width,
                currentInstance.getImputSetUp().heigth);
        currentInstance.inputField.setBackground(Color.GREEN);// Background
                                                                // colour
        currentInstance.inputField.setForeground(Color.BLUE);// font colour
        currentInstance.inputField.setFont(currentInstance.TEXT_FONT); // sets
                                                                        // font
        currentInstance.inputField.setVisible(true);// visibility
        currentInstance.inputField.setEnabled(true);// editability
        currentInstance.inputField.setColumns(10);// width
        currentInstance.inputField.addMouseMotionListener(this);// motion
```

```java
                                                                    // sensitive
            currentInstance.inputField.addActionListener(this);// enter-press
                                                                // sensitive
            contentPane.add(currentInstance.inputField);// adds it to content
                                                        // pane
            answerTries = 0;// resets anwer tries
        }
    }

    private boolean checkImput(JTextField imputField, Variables instance) {
        String imput = imputField.getText();// Fetches input from input field
        if (imput.length() == 0) {// If there's no input
            imputField.setBackground(new Color(16744448));// changes colour to
                                                            // "error-colour".
            return false;// and returns false
        }
        int i = getAnwerSheet(instance);// gets corresponding answer sheet
        for (int j = 0; j < 5; j++) {// checks all the possible answers of
                                        // answer sheet
            if (imput.equals(Variables.answers[i][j])) {
                return true; // returns true, if match is found, method ends
                                // here.
            }
        }
        imputField.setBackground(new Color(16744448));// if method did not end,
                                                        // there is no match and
                                                        // colour is changed to
                                                        // "error-colour".
        if (answerTries >= 5) {// Displays hint after five tries.
            instance.button[instance.getButtonCount() - 1].setVisible(true);// sets
                                                                            // hint
                                                                            //
button
                                                                            //
visible
            refresh();
        }
        answerTries++;// increments answer tries
        return false;
    }

    private void setButtonSelected(JButton button) {// highlights choice button,
                                                    // on mouse hoover.
        button.setFont(Variables.CHOICE_FONT_SELECTED);
        button.setForeground(Color.BLUE);
        refresh();
    }

    private void setButtonUnselected(Variables currentInstance) {// Unhighlights
                                                                    // button on
                                                                    // mouse
                                                                    // hoover.
        for (int i = 0; i < currentInstance.getButtonCount(); i++) {
            if (currentInstance.button[i].getFont() != null) {
                currentInstance.button[i].setFont(currentInstance
                        .getButtonSetUp(i).font);
                currentInstance.button[i].setForeground(Color.WHITE);
            }
            refresh();
        }
    }

    private void refresh() {// used to redisplay components of the frame.
        revalidate();
        repaint();
    }

    private void nextSlide() {// Whenever slide is changed, this method is
                              // called.
        letterTimer.stop();// Before starting the animation of new text, old
                           // animation has to be stopped.
        loadUpContent(getCurrentInstance());// Loads up content.
        if (getCurrentInstance().getTextSetUp() != null) {// If there is Text on
                                                          // a slide, starts
                                                          // the animation
            readLine(getCurrentInstance()); // by calling readLine
        }
        // System.out.println(getCurrentInstance());//used only when diagnosing
        // the program
    }

    @Override
    public void actionPerformed(ActionEvent somethingHappened) {/**/// Timer
                                                                  // evens
                                                                  // and
                                                                  //
"enter-press"
                                                                  // event
                                                                  // are
                                                                  // handled
                                                                  // here
        // TODO Auto-generated method stub
        Variables currentInstance = getCurrentInstance();// Following code
        // always need to
        // work only for
        // instance of
        // currently
        // displayed slide.
        // This slide is
        // determined here
        // by calling
        // getCurrentSlide.
        if (somethingHappened.getSource() == letterTimer) {// If there still is
                                                           // a line to be
                                                           // shown (that means
                                                           // the line is not
                                                           // null).
            if (line == null) {
            } else
                addLetter(currentInstance);// adds 1 letter (or a new line sign)
                                           // to the text displayed.

        } else if (somethingHappened.getSource() == buttonDisplayDelayTimer) {//
after
                                                                              //
little
                                                                              //
delay,
                                                                              //
displays
                                                                              //
all
                                                                              //
the
                                                                              //
buttons
                                                                              //
except
```

```
                                                                            //
    following
                                                                            //
    {so
                                                                            //
    basically
                                                                            //
    only
                                                                            //
    "next"
                                                                            //
    button):
            for (int i = 0; i <= currentInstance.getButtonCount() - 1; i++) {
                if (currentInstance.getButtonSetUp(i).caption.equals(">> exit") !=
    true // the
    
    / exit
    
    / button
                        && currentInstance.getButtonSetUp(i).icon == null // the
                                                                         //
    button
                                                                         // with
                                                                         // icon
                                                                         //
    (red-arrow
                                                                         //
    button)
                        && currentInstance.getHint() == null) {// hint button
                    currentInstance.button(i).setVisible(true);
                }
            }
            buttonDisplayDelayTimer.stop();// button display timer can now be
                                            // stopped.
        } else if (somethingHappened.getSource() == currentInstance.inputField) {//
    this
    
    / handles
    
    / the
    
    / event
    
    / fired
    
    / from
    
    / input
    
    / field
    
    / by
    
    / pressing
    
    / enter.
            if (checkImput(getCurrentInstance().inputField,// if the answer is
                                                           // correct, displays
                                                           // next slide.
                    getCurrentInstance())) {
                currentSlide++;
                nextSlide();
            }
        }
```

```
}

@Override
public void mouseClicked(MouseEvent event) {/**/// This method controls the
    // program flow. Depending on
    // the clicked button, different
    // slides will be displayed
    // next.
    // TODO Auto-generated method stub
    Variables currentInstance = getCurrentInstance();// Following code
    // always needs to
    // work only for
    // instance of
    // currently
    // displayed slide.
    // This slide is
    // determined here
    // by calling
    // getCurrentSlide.

    if (event.getSource() == currentInstance.button[0]) {// Button with
                                                            // index 0 is an
                                                            // exit button.
                                                            // It is on
                                                            // every slide,
                                                            // although it
                                                            // is hidden on
                                                            // most of
                                                            // them..
        System.exit(0);// exits app.
    } else if (event.getSource() == ae2Instance.button[1]) {// Button on a
                                                            // selection
                                                            // page. When
                                                            // option 2 is
                                                            // chosen, some
                                                            // slides need
                                                            // to be skipped
        currentSlide = 9;
        nextSlide();
    } else if (event.getSource() == azInstance.button[2]) {// Same as
                                                            // previous.
        currentSlide = 36;
        nextSlide();
    } else if (event.getSource() == adInstance.button[2]) {// Same as
                                                            // previous.
        currentSlide = 7;
        nextSlide();

    } else if (event.getSource() == awInstance.button[1]// "Enter" buttons
                                                        // on slides with
                                                        // puzzles
            || event.getSource() == bd1Instance.button[1]// The answer is
                                                            // checked
                                                            // before moving
                                                            // to next slide
            || event.getSource() == bd2Instance.button[1]) {
        if (checkImput(getCurrentInstance().inputField,// Checks the answer
                getCurrentInstance())) {
            currentSlide++;
            nextSlide();
        }
    } else if (

    event.getSource() == currentInstance.button[1]// button one on every
```

```
                                                        // slide is "next"
                                                        // button.
                || event.getSource() == getCurrentInstance().button[1]) {//
clicking
                                                                            // it
                                                                            // will
                                                                            // move
                                                                            // to
                                                                            // next
                                                                            //
slide
            currentSlide++;
            nextSlide();

        } else if (event.getSource() == abInstance.button[2]// red arrow
                                                            // responsible for
                                                            // displaying next
                                                            // part of tet on
                                                            // same slide.
                || event.getSource() == agInstance.button[2]
                || event.getSource() == bc1Instance.button[2]) {
            letterTimer.setInitialDelay(0);
            readLine(getCurrentInstance());
        }
    }

    @Override
    public void mouseMoved(MouseEvent event) {/**/// Used to change cursor, when
                                                    // hoovered over object.
        Variables currentInstance = getCurrentInstance(); // Following code
                                                            // always need to
                                                            // work only for
                                                            // instance of
                                                            // currently
                                                            // displayed slide.
                                                            // This slide is
                                                            // determined here
                                                            // by calling
                                                            // getCurrentSlide.

        // TODO Auto-generated method stub
        if (event.getSource() == this) {// When mouse hoovered anywhere inside
                                        // the frame (that means anywhere but
                                        // the buttons), cursor is changed back
                                        // to normal.
            this.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));//
sets
                                                                                //
cursor
            setButtonUnselected(currentInstance);// by calling
                                                    // setButtonUnselected
                                                    // removes the
                                                    // selection(highlight) of
                                                    // any button.

        } else if (event.getSource() == adInstance.button[1]
                || event.getSource() == azInstance.button[1]
                || event.getSource() == adInstance.button[2]
                || event.getSource() == azInstance.button[2]) {
            this.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));// sets
                                                                            // hand
                                                                            //
cursor
            setButtonSelected((JButton) event.getSource());// Selects/highlights
```

```
                                                                 // the button.

        } else if (event.getSource() == currentInstance.button[0]// This means
                                                                    // that for
                || event.getSource() == currentInstance.button[1]// any button
                                                                    // on any
                                                                    // slide
                || event.getSource() == currentInstance.button[2]// cursor will
                                                                    // change to
                                                                    // hand
                || event.getSource() == currentInstance.button[3]) {
            this.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));

        } else if (event.getSource() == awInstance.inputField // This does not
                || event.getSource() == bd1Instance.inputField// change the
                                                                // cursor.
                || event.getSource() == bd2Instance.inputField) {// but removes
                                                                    // the red
            currentInstance.inputField.setBackground(Color.GREEN);// background
                                                                    // (red
                                                                    // background
                                                                    // is added,
                                                                    // when user
                                                                    // enters
                                                                    // wrong
                                                                    // anwer.
            refresh();
        }
    }

    @Override
    public void mouseDragged(MouseEvent e) {/**/// Following methods are not
                                                    // used,
        // but have to be present, because
        // of implementation of mouse and
        // action listeners.
        // TODO Auto-generated method stub

    }

    @Override
    public void mouseExited(MouseEvent e) {/**/// Not used
        // TODO Auto-generated method stub

    }

    @Override
    public void mousePressed(MouseEvent e) {/**/// Not used
        // TODO Auto-generated method stub

    }

    @Override
    public void mouseReleased(MouseEvent e) {/**/// Not used
        // TODO Auto-generated method stub

    }

    @Override
    public void mouseEntered(MouseEvent e) {/**/// Not used
        // TODO Auto-generated method stub

    }
```

```
}

```