

MainWindow.java

```
1 package firstGraph;
2
3 import java.awt.Color;
4 import java.awt.Cursor;
5 import java.awt.Dimension;
6 import java.awt.Font;
7 import java.awt.Image;
8 import java.awt.Toolkit;
9 import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11 import java.awt.event.MouseEvent;
12 import java.awt.event.MouseListener;
13 import java.awt.event.MouseMotionListener;
14 import java.io.BufferedReader;
15 import java.io.File;
16 import java.io.FileReader;
17 import java.io.IOException;
18
19 import javax.imageio.ImageIO;
20 import javax.swing.ImageIcon;
21 import javax.swing.JButton;
22 import javax.swing.JFrame;
23 import javax.swing.JLabel;
24 import javax.swing.JPanel;
25 import javax.swing.JTextArea;
26 import javax.swing.JTextField;
27 import javax.swing.SwingConstants;
28 import javax.swing.Timer;
29
30 @SuppressWarnings("serial")
31 public class MainWindow extends JFrame implements ActionListener,
32     MouseListener, MouseMotionListener {
33
34     public int currentSlide = 1; // Number of slide that is loaded at launch
35
36     AA_FirstSlide aaInstance; // Each slide has its own class.
37     AB_SecondSlide abInstance; // Each class is placed in separate file.
38     AC_ThirdSlide acInstance; // All the classes are sub-classes of
39     AD_FourthSlide adInstance; // Variables class.
40     AE1_FifthSlide ae1Instance; // These classes are declared here.
41     AE2_FifthSlide ae2Instance;
42     AF1_SixthSlide af1Instance;
43     AF2_SixthSlide af2Instance;
44     AG_SeventhSlide agInstance;
45     AH_EightSlide ahInstance;
46     AI_NinthSlide aiInstance;
47     AJ_TenthSlide ajInstance;
48     AK_EleventhSlide akInstance;
49     AL_TwelfthSlide alInstance;
50     AM_ThirteenthSlide amInstance;
51     AN_FourteenthSlide anInstance;
52     AO_FifteenthSlide aoInstance;
53     AP_SixteenthSlide apInstance;
54     AQ_SeventeenthSlide aqInstance;
55     AR_EighteenthSlide arInstance;
56     AS_NinteenthSlide asInstance;
57     AT_TwentiethSlide atInstance;
58     AU_21stSlide auInstance;
59     AV_22ndSlide avInstance;
60     AW_23rdSlide awInstance;
61     AX_24thSlide axInstance;
62     AY_25thSlide ayInstance;
63     AZ_26thSlide azInstance;
64     BA1_27thSlide ba1Instance;
```

# MainWindow.java

```

65     BB1_28thSlide bblInstance;
66     BC1_29thSlide bclInstance;
67     BD1_30thSlide bdlInstance;
68     BE1_31stSlide belInstance;
69     BF1_32ndSlide bflInstance;
70     BG1_33rdSlide bglInstance;
71     BA2_27thSlide ba2Instance;
72     BB2_28thSlide bb2Instance;
73     BC2_29thSlide bc2Instance;
74     BD2_30thSlide bd2Instance;
75     BE2_31stSlide be2Instance;
76     BF2_32ndSlide bf2Instance;
77
78     String line; // Used when displaying current line to the JTextArea
79     int currentChar = 0; // Lastly displayed character.
80     int currentLine = 0; // Line currently being displayed
81     int totalLines; // Total lines in the text file.
82     int answerTries; // Counts the answer tries. After 5 displays hint.
83
84     Timer letterTimer; // Timer responsible for displaying letters one by one.
85     int letterTimerDelay = 1000;
86
87     Timer buttonDisplayDelayTimer; // Timer responsible for displaying buttons
88     int buttonTimerDelay = 2000; // with delay.
89
90     JPanel contentPane; // Content pane holding the buttons and the textArea.
91
92     public MainWindow() /**/ // Constructor.
93     {
94         setUpFrame(); // Method used to display the window.
95         initiateInstances(); // This method must be run AFTER the setUpFrame
96                             // method, because when constructing each class, the
97                             // screen width and height are used.
98         loadUpContent getCurrentInstance(); // Loads up the actual content of
99                                         // current slide.
100    }
101
102    private void initiateInstances() // Creates instance of each slide-class
103    { // and both timers.
104        aaInstance = new AA_FirstSlide();
105        abInstance = new AB_SecondSlide();
106        acInstance = new AC_ThirdSlide();
107        adInstance = new AD_FourthSlide();
108        ae1Instance = new AE1_FifthSlide();
109        aflInstance = new AF1_SixthSlide();
110        ae2Instance = new AE2_FifthSlide();
111        af2Instance = new AF2_SixthSlide();
112        agInstance = new AG_SeventhSlide();
113        ahInstance = new AH_EightSlide();
114        aiInstance = new AI_NinthSlide();
115        ajInstance = new AJ_TenthSlide();
116        akInstance = new AK_EleventhSlide();
117        alInstance = new AL_TwelfthSlide();
118        amInstance = new AM_ThirteenthSlide();
119        anInstance = new AN_FourteenthSlide();
120        aoInstance = new AO_FifteenthSlide();
121        apInstance = new AP_SixteenthSlide();
122        aqInstance = new AQ_SeventeenthSlide();
123        arInstance = new AR_EighteenthSlide();
124        asInstance = new AS_NinteenthSlide();
125        atInstance = new AT_TwentiethSlide();
126        auInstance = new AU_21stSlide();
127        avInstance = new AV_22ndSlide();
128        awInstance = new AW_23rdSlide();
129        axInstance = new AX_24thSlide();

```

# MainWindow.java

```

129     ayInstance = new AY_25thSlide();
130     azInstance = new AZ_26thSlide();
131     ba1Instance = new BA1_27thSlide();
132     bb1Instance = new BB1_28thSlide();
133     bc1Instance = new BC1_29thSlide();
134     bd1Instance = new BD1_30thSlide();
135     be1Instance = new BE1_31stSlide();
136     bf1Instance = new BF1_32ndSlide();
137     bg1Instance = new BG1_33rdSlide();
138     ba2Instance = new BA2_27thSlide();
139     bb2Instance = new BB2_28thSlide();
140     bc2Instance = new BC2_29thSlide();
141     bd2Instance = new BD2_30thSlide();
142     be2Instance = new BE2_31stSlide();
143     bf2Instance = new BF2_32ndSlide();
144
145     letterTimer = new Timer(getCurrentInstance().TIMER_SPEED,
146                             MainWindow.this);
147     buttonDisplayDelayTimer = new Timer(200, MainWindow.this);
148 }
149
150 private Variables getCurrentInstance() // Returns correct class instance
151                                     // depending on slide currently
152                                     // displayed.
153                                     // Most used method through whole
154                                     // program. Used in every method,
155                                     // that is shared by multiple
156                                     // slides.
157
158     switch (currentSlide) {
159     case 1:
160         return aaInstance;
161     case 2:
162         return abInstance;
163     case 3:
164         return acInstance;
165     case 4:
166         return adInstance;
167     case 5:
168         return ae1Instance;
169     case 6:
170         return ae2Instance;
171     case 7:
172         return af1Instance;
173     case 8:
174         return af2Instance;
175     case 9:
176         return agInstance;
177     case 10:
178         return ahInstance;
179     case 11:
180         return aiInstance;
181     case 12:
182         return ajInstance;
183     case 13:
184         return akInstance;
185     case 14:
186         return alInstance;
187     case 15:
188         return amInstance;
189     case 16:
190         return anInstance;
191     case 17:
192         return aoInstance;
193     case 18:

```

```

193         return apInstance;
194     case 19:
195         return aqInstance;
196     case 20:
197         return arInstance;
198     case 21:
199         return asInstance;
200     case 22:
201         return atInstance;
202     case 23:
203         return auInstance;
204     case 24:
205         return avInstance;
206     case 25:
207         return awInstance;
208     case 26:
209         return axInstance;
210     case 27:
211         return ayInstance;
212     case 28:
213         return azInstance;
214     case 29:
215         return balInstance;
216     case 30:
217         return bblInstance;
218     case 31:
219         return bclInstance;
220     case 32:
221         return bdlInstance;
222     case 33:
223         return belInstance;
224     case 34:
225         return bflInstance;
226     case 35:
227         return bg1Instance;
228     case 36:
229         return ba2Instance;
230     case 37:
231         return bb2Instance;
232     case 38:
233         return bc2Instance;
234     case 39:
235         return bd2Instance;
236     case 40:
237         return be2Instance;
238     case 41:
239         return bf2Instance;
240
241     default:
242         System.out.println("Swear word, that was"
243             + " here used for testing purposes "
244             + "was replaced before hand-in.");
245         return null;
246     }
247 }
248
249 private int getAnswerSheet Variables instance) { // Returns answer sheet
250 // corresponding to the
251 // slide with question.
252     if (instance == awInstance) {
253         return 0;
254     } else if (instance == bdlInstance) {
255         return 1;
256     } else if (instance == bd2Instance) {

```

```

257         return 2;
258     } else
259         return -1;
260 }
261
262 private void setUpFrame() { // This method is called only once (when program
263                             // is launched).
264
265     setUndecorated true ; // No border around the window (to be fullscreen).
266     setExtendedState JFrame.MAXIMIZED_BOTH ;// Fullscreen
267     setSize new Dimension Toolkit.getDefaultToolkit().getScreenSize(); //
Fullscreen
268     setResizable false ; // not resizable
269     setVisible true ; // Visible
270     setDefaultCloseOperation JFrame.EXIT_ON_CLOSE ;// Application is
271                                     // terminated after
272                                     // closing.
273     setBackground Color.BLACK ; // background color
274     Main.dimX = getWidth(); // Assigning screen width and height to the
275                             // variable, which
276     Main.dimY = getHeight(); // is used elsewhere in program.
277     Variables.DEFAULT_BUTTON_WIDTH = Variables.calculatePos(9, true); // Sets
278                                     // the
279                                     //
default
280                                     //
button
281                                     // size
282                                     //
depending
283                                     // on
284                                     //
screen
285                                     // res.
286     Variables.DEFAULT_BUTTON_HEIGHT = Variables.calculatePos(2, false);
287     if Main.dimX <= 1200 || Main.dimY <= 700 { // If screen is small,
288                                     // makes font smaller.
289         Variables.BUTTON_FONT_SIZE = 20;
290         Variables.TEXT_FONT_SIZE = 24;
291     }
292     addMouseListener this ; // Motion listener used to change cursor
293                             // back to default (after changing to
294                             // hand cursor.
295 }
296
297 private void loadUpContent Variables currentInstance { // Loads up the
298                                     // background image.
299                                     // Then load
300     // the content of content pane (buttons,
301     // text...)
302     loadImage currentInstance ; // Method used for loading the image
303     setUpContentPane currentInstance ;// Method used for setting up the
304                                     // content pane.
305     // Method creating buttons is called from this
306     // method.
307     setUpTextArea currentInstance ;// Method used for setting up text
308                                     // area (it there is one).
309     setUpInputField currentInstance ; // Method used for creating input
310                                     // field (if there is one).
311     contentPane.add(currentInstance.label);
312     refresh(); // Recalculating and repainting the frame.
313     letterTimer.setInitialDelay(letterTimerDelay); // Setting the waiting
314     buttonDisplayDelayTimer.setInitialDelay(buttonTimerDelay); // time,
315                                     // before

```

```

316                                     // timer
317     // action is run.
318
319 }
320
321 private void loadImage Variables currentInstance) { // Method loading the
322                                     // background image
323     try { // Following needs to be surrounded in try-catch in case the file
324           // was corrupted (for example).
325         currentInstance.image1 = ImageIO.read new File(currentInstance
326             .getPIC_PATH()); // Reads pic to file
327         currentInstance.myImageIcon = new ImageIcon( // "Transforms" file to
328             // displayable icon,
329             currentInstance.image1.getScaledInstance(-1, Main.dimY, //
Resizes
330                                     // icon
331                                     // to
332                                     // fit
333                                     // the
334                                     //
screen.
335                                     Image.SCALE_SMOOTH);
336     } catch IOException e {
337         System.err.println("Error loading picture.");
338         System.exit(0);
339     } catch NullPointerException e {
340         System.err.println("Error loading picture NPE.");
341         System.exit(0);
342     }
343
344     currentInstance.label = new JLabel(); // Creates a label
345     currentInstance.label.setIcon(currentInstance.myImageIcon); // Attaches
346                                     // the icon
347                                     // to this
348                                     // label
349     currentInstance.label.setBounds(
350         // And places the label to fit the screen.
351         (Main.dimX - currentInstance.myImageIcon.getIconWidth()) / 2,
352         0, Main.dimX, Main.dimY);
353 }
354
355 private void setUpContentPane Variables currentInstance) { // Sets up
356                                     // content pane
357                                     // (the "glass"
358                                     // of the
359                                     // window)
360     contentPane = new JPanel(); // Create
361     contentPane.setBackground Color.BLACK; // Background
362     contentPane.setBorder null; // border
363     setContentPane contentPane);
364     contentPane.setLayout null; // sets no layout (pre-defined layouts are
365                                     // unusable for us).
366
367     for int i = 0; i < currentInstance.getButtonCount(); i++) {
368         setUpButton(
369             currentInstance,
370             i, // Loop that calls the method creating buttons.
371             currentInstance.getButtonSetUp(i).posX, // It uses the
372                                     // properties from
373                                     // current slide
374                                     // class.
375             currentInstance.getButtonSetUp(i).posY, // It passes them to
376                                     // the setUpButton
377                                     // method as

```

# MainWindow.java

```

378                                     // parameters.
379         currentInstance.getButtonSetUp(i).width,
380         currentInstance.getButtonSetUp(i).height,
381         currentInstance.getButtonSetUp(i).caption,
382         currentInstance.getButtonSetUp(i).font,
383         currentInstance.getButtonSetUp(i).icon,
384         currentInstance.getButtonSetUp(i).visible);
385     }
386 }
387
388 private void setUpButton Variables currentInstance, int id, int positionX,
389         int positionY, int width, int height, String name, Font font,
390         String icon, boolean visible) // Method for creating buttons with
391 // button properties as parameters.
392     currentInstance.button[id] = new JButton(name); // creates a button
393     currentInstance.button[id].setBorder(null); // border
394     currentInstance.button[id].setHorizontalAlignment(SwingConstants.LEFT); //
text
395                                     //
inside
396                                     //
the
397                                     //
button
398                                     //
alignment.
399     currentInstance.button[id].setContentAreaFilled(false); // the button is
400                                     // transparent
401     currentInstance.button[id].setOpaque(false); // transparency
402     currentInstance.button[id].setForeground(Color.WHITE); // font color
403     if (font != null) // some buttons have only icon, hence no font
404         currentInstance.button[id].setFont(font);
405 }
406 if (icon != null) // some buttons dont have icon...
407     currentInstance.button[id].setIcon(new ImageIcon(icon));
408 }
409 if (visible == false) { // some buttons are not visible from the
410     // beginning
411     currentInstance.button[id].setVisible(false);
412 }
413 currentInstance.button[id].setBounds(positionX, positionY, width,
414     height); // sets button position
415
416 currentInstance.button[id].addMouseListener(this); // makes button
417     // click-sensitive
418 currentInstance.button[id].addMouseMotionListener(this); // makes button
419     // motion-sensitive
420
421 contentPane.add(currentInstance.button[id]); // finally, adds button to
422     // the content pane
423 }
424
425 private void setUpTextArea Variables currentInstance) // sets up text area
426     if (currentInstance.getTextSetUp() != null) { // if there is no text
427     // area defined in slide
428     // class, nothing
429     // happens.
430     currentInstance.textArea = new JTextArea(); // creates text
431     // Area
432     currentInstance.textArea.setBounds(
433     // positions the tet area
434     currentInstance.getTextSetUp().posX,
435     currentInstance.getTextSetUp().posY,
436     currentInstance.getTextSetUp().width,

```

# MainWindow.java

```

437         currentInstance.getTextSetUp().height);
438
439         currentInstance.textArea.setWrapStyleWord(true); // breaks the text
440                                                         // into multiple
441                                                         // lines
442         currentInstance.textArea.setLineWrap(true);
443         currentInstance.textArea.setEditable(false); // is not editable
444         currentInstance.textArea.setText(currentInstance.text); // sets text
445
446         if (currentInstance.getTextSetUp().someColor != null) // if the
447                                                         // color is
448                                                         // specified,
449                                                         // uses the
450                                                         // color.
451             currentInstance.textArea.setForeground(currentInstance
452                 .getTextSetUp().someColor);
453     } else
454         // Otherwise uses default color.
455         currentInstance.textArea
456             .setForeground(Variables.DEF_TEXT_COLOR);
457
458     if (currentInstance == abInstance) { // Only second slide has Red
459                                         // text color.
460         currentInstance.textArea.setBackground(new Color(float 0.37,
461             float 0.37, float 0.37, float 0.7));
462     } else // all other slides have default text color
463         currentInstance.textArea
464             .setBackground(Variables.DEF_TEXTAREA_COLOR);
465     }
466     currentInstance.textArea // sets the defined font.
467         .setFont(currentInstance.getTextSetUp().font);
468
469     contentPane.add(currentInstance.textArea); // Adds text area to
470                                               // content pane.
471
472     openFile(getCurrentInstance()); // opens a file with text
473 }
474
475
476 private void openFile(Variables currentInstance) // opens file
477     try // Following needs to be surrounded in try-catch in case the file
478         // was corrupted (for example).
479         currentInstance.myFile = new File(currentInstance.getFILE_PATH());
480         currentInstance.filRead = new FileReader(currentInstance.myFile); //
sets
481                                                         //
file
482                                                         //
reader
483         currentInstance.bufRead = new BufferedReader(// buffered reader
484             currentInstance.filRead);
485         totalLines = Integer.parseInt(currentInstance.bufRead.readLine()); //
the
486                                                         //
first
487                                                         //
line
488                                                         //
in
489                                                         //
file
490                                                         //
has
491                                                         //

```



```

to
492                                                                    //
be
493                                                                    //
a
494                                                                    //
number
495                                                                    //
with
496                                                                    //
total
497                                                                    //
lines.
498                                                                    //
Reads
499                                                                    //
the
500                                                                    //
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;

```

```

546         buttonDisplayDelayTimer.start();
547     }
548     catch (NullPointerException exception) {
549     }
550
551     currentChar = 0; // resets line counter
552     currentInstance.text = ""; // wipes the content of text area, so the
553                             // animation of adding text letter-by-letter
554                             // can be played
555     letterTimer.start(); // starts the animation.
556 }
557
558 private void addLetter Variables currentInstance) // adds one letter -
559                                     // animation
560     if (currentChar >= line.length()) // keeps adding letters until all
561                                     // line is displayed
562         letterTimer.stop();
563     else {
564         if (line.charAt(currentChar) == '\n') // special char in text files
565                                             // used to break text into
566                                             // multiple lines
567             currentInstance.text = currentInstance.text + "\n"; // adds
568                                                         // breakpoint
569                                                         // to
570                                                         // string.
571         else {
572             currentInstance.text = currentInstance.text
573                 + line.charAt(currentChar); // adds letter to string
574         }
575
576         currentChar++;
577         currentInstance.textArea.setText(currentInstance.text); // sets the
578                                                         // changed
579                                                         // string as
580                                                         // text area
581                                                         // text
582         currentInstance.textArea.revalidate(); // Recalculates the text
583                                                         // area.
584         refresh();
585     }
586 }
587
588 private void setUpInputField Variables currentInstance) // sets up input
589                                     // field
590     if (currentInstance.getInputSetUp() != null) // if there is not input
591                                     // field specified,
592                                     // nothing happens
593         currentInstance.inputField = new JTextField(); // creates the input
594                                     // field
595     currentInstance.inputField.setBounds(
596         // positions it.
597         currentInstance.getInputSetUp().posX,
598         currentInstance.getInputSetUp().posY,
599         currentInstance.getInputSetUp().width,
600         currentInstance.getInputSetUp().height);
601     currentInstance.inputField.setBackground Color.GREEN; // Background
602                                     // colour
603     currentInstance.inputField.setForeground Color.BLUE; // font colour
604     currentInstance.inputField.setFont(currentInstance.TEXT_FONT); // sets
605                                     // font
606     currentInstance.inputField.setVisible true; // visibility
607     currentInstance.inputField.setEnabled true; // editability
608     currentInstance.inputField.setColumns(10); // width
609     currentInstance.inputField.addMouseMotionListener(this); // motion

```

```

610                                     // sensitive
611     currentInstance.inputField.addActionListener(this); // enter-press
612                                     // sensitive
613     contentPane.add(currentInstance.inputField); // adds it to content
614                                     // pane
615     answerTries = 0; // resets anwer tries
616 }
617
618
619 private boolean checkInput(JTextField inputField, Variables instance) {
620     String input = inputField.getText(); // Fetches input from input field
621     if (input.length() == 0) // If there's no input
622         inputField.setBackground(new Color(16744448)); // changes colour to
623                                                         // "error-colour".
624     return false; // and returns false
625 }
626
627 int i = getAnswerSheet(instance); // gets corresponding answer sheet
628 for (int j = 0; j < 5; j++) // checks all the possible answers of
629                             // answer sheet
630     if (input.equals(Variables.answers[i][j])) {
631         return true; // returns true, if match is found, method ends
632                     // here.
633     }
634     inputField.setBackground(new Color(16744448)); // if method did not end,
635                                                         // there is no match and
636                                                         // colour is changed to
637                                                         // "error-colour".
638     if (answerTries >= 5) // Displays hint after five tries.
639         instance.button[instance.getButtonCount() - 1].setVisible(true); // sets
640                                                         // hint
641                                                         //
642     button                                                         //
643     visible                                                         //
644         refresh();
645     answerTries++; // increments answer tries
646     return false;
647 }
648
649 private void setButtonSelected(JButton button) // highlights choice button,
650                                               // on mouse hoover.
651     button.setFont(Variables.CHOICE_FONT_SELECTED);
652     button.setForeground(Color.BLUE);
653     refresh();
654 }
655
656 private void setButtonUnselected(Variables currentInstance) // Unhighlights
657                                                         // button on
658                                                         // mouse
659                                                         // hoover.
660     for (int i = 0; i < currentInstance.getButtonCount(); i++) {
661         if (currentInstance.button[i].getFont() != null) {
662             currentInstance.button[i].setFont(currentInstance
663                 .getButtonSetUp(i).font);
664             currentInstance.button[i].setForeground(Color.WHITE);
665         }
666         refresh();
667     }
668 }
669
670 private void refresh() // used to redisplay components of the frame.
671     revalidate();

```

```

672     repaint();
673 }
674
675 private void nextSlide() // Whenever slide is changed, this method is
676                        // called.
677     letterTimer.stop(); // Before starting the animation of new text, old
678                        // animation has to be stopped.
679     loadUpContent(getCurrentInstance()); // Loads up content.
680     if (getCurrentInstance().getTextSetUp() != null) // If there is Text on
681                                                    // a slide, starts
682                                                    // the animation
683         readLine(getCurrentInstance()); // by calling readLine
684 }
685 // System.out.println(getCurrentInstance()); // used only when diagnosing
686 // the program
687 }
688
689 @Override
690 public void actionPerformed(ActionEvent somethingHappened) /**/// Timer
691                                                    // evens
692                                                    // and
693                                                    //
694 "enter-press"
695                                                    // event
696                                                    // are
697                                                    // handled
698                                                    // here
699 // TODO Auto-generated method stub
700 Variables currentInstance = getCurrentInstance(); // Following code
701 // always need to
702 // work only for
703 // instance of
704 // currently
705 // displayed slide.
706 // This slide is
707 // determined here
708 // by calling
709 // getCurrentSlide.
710 if (somethingHappened.getSource() == letterTimer) // If there still is
711                                                    // a line to be
712                                                    // shown (that means
713                                                    // the line is not
714                                                    // null).
715     if (line == null) {
716     } else
717         addLetter(currentInstance); // adds 1 letter (or a new line sign)
718                                     // to the text displayed.
719     else if (somethingHappened.getSource() == buttonDisplayDelayTimer) //
720 after
721 little
722 delay,
723 displays
724 all
725 the
726 buttons
727 except

```

# MainWindow.java

```

727                                                                 //
    following
728                                                                 //
    {so
729                                                                 //
    basically
730                                                                 //
    only
731                                                                 //
    "next"
732                                                                 //
    button):
733         for int i = 0; i <= currentInstance.getButtonCount() - 1; i++ {
734             if (currentInstance.getButtonSetUp(i).caption.equals(">> exit") !=
true // the
735                 / exit
736                 / button
737                 && currentInstance.getButtonSetUp(i).icon == null // the
738                                     //
button
739                                     // with
740                                     // icon
741                                     //
(red-arrow
742                                     //
button)
743                 && currentInstance.getHint() == null) { // hint button
744                     currentInstance.button[i].setVisible(true);
745                 }
746             }
747             buttonDisplayDelayTimer.stop(); // button display timer can now be
748                                     // stopped.
749         } else if (somethingHappened.getSource() == currentInstance.inputField) { //
this
750         / handles
751         / the
752         / event
753         / fired
754         / from
755         / input
756         / field
757         / by
758         / pressing
759         / enter.
760         if (checkInput(currentInstance().inputField, // if the answer is
761                                     // correct, displays
762                                     // next slide.
763                                     currentInstance())) {
764             currentSlide++;
765             nextSlide();
766         }
767     }

```

```

768 }
769
770 @Override
771 public void mouseClicked(MouseEvent event) (**/// This method controls the
772     // program flow. Depending on
773     // the clicked button, different
774     // slides will be displayed
775     // next.
776     // TODO Auto-generated method stub
777     Variables currentInstance = getCurrentInstance(); // Following code
778     // always needs to
779     // work only for
780     // instance of
781     // currently
782     // displayed slide.
783     // This slide is
784     // determined here
785     // by calling
786     // getCurrentSlide.
787
788     if (event.getSource() == currentInstance.button[0]) (// Button with
789                                                         // index 0 is an
790                                                         // exit button.
791                                                         // It is on
792                                                         // every slide,
793                                                         // although it
794                                                         // is hidden on
795                                                         // most of
796                                                         // them..
797
798         System.exit(0); // exits app.
799     } else if (event.getSource() == ae2Instance.button[1]) (// Button on a
800                                                         // selection
801                                                         // page. When
802                                                         // option 2 is
803                                                         // chosen, some
804                                                         // slides need
805                                                         // to be skipped
806
807         currentSlide = 9;
808         nextSlide();
809     } else if (event.getSource() == azInstance.button[2]) (// Same as
810                                                         // previous.
811
812         currentSlide = 36;
813         nextSlide();
814     } else if (event.getSource() == adInstance.button[2]) (// Same as
815                                                         // previous.
816
817         currentSlide = 7;
818         nextSlide();
819     } else if (event.getSource() == awInstance.button[1]) (// "Enter" buttons
820                                                         // on slides with
821                                                         // puzzles
822
823         || event.getSource() == bd1Instance.button[1] (// The answer is
824                                                         // checked
825                                                         // before moving
826                                                         // to next slide
827
828         || event.getSource() == bd2Instance.button[1]) (
829         if (checkInput(getCurrentInstance().inputField, // Checks the answer
830                     getCurrentInstance())) {
831             currentSlide++;
832             nextSlide();
833         }
834     } else if (
835
836 event.getSource() == currentInstance.button[1] // button one on every

```

# MainWindow.java

```

832                                     // slide is "next"
833                                     // button.
834         || event.getSource() == getCurrentInstance().button[1] //
clicking
835                                     // it
836                                     // will
837                                     // move
838                                     // to
839                                     // next
840                                     //
slide
841         currentSlide++;
842         nextSlide();
843
844     } else if (event.getSource() == abInstance.button[2] // red arrow
845                                     // responsible for
846                                     // displaying next
847                                     // part of tet on
848                                     // same slide.
849         || event.getSource() == agInstance.button[2]
850         || event.getSource() == bclInstance.button[2]) {
851         letterTimer.setInitialDelay(0);
852         readLine(getCurrentInstance());
853     }
854 }
855
856 @Override
857 public void mouseMoved(MouseEvent event) /**/// Used to change cursor, when
858                                     // hovered over object.
859     Variables currentInstance = getCurrentInstance(); // Following code
860                                     // always need to
861                                     // work only for
862                                     // instance of
863                                     // currently
864                                     // displayed slide.
865                                     // This slide is
866                                     // determined here
867                                     // by calling
868                                     // getCurrentSlide.
869
870 // TODO Auto-generated method stub
871 if (event.getSource() == this) // When mouse hovered anywhere inside
872                                     // the frame (that means anywhere but
873                                     // the buttons), cursor is changed back
874                                     // to normal.
875     this.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR)); //
sets
876                                     //
cursor
877     setButtonUnselected(currentInstance); // by calling
878                                     // setButtonUnselected
879                                     // removes the
880                                     // selection(highlight) of
881                                     // any button.
882
883     } else if (event.getSource() == adInstance.button[1]
884         || event.getSource() == azInstance.button[1]
885         || event.getSource() == adInstance.button[2]
886         || event.getSource() == azInstance.button[2]) {
887     this.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR)); // sets
888                                     // hand
889                                     //
cursor
890     setButtonSelected((JButton) event.getSource()); // Selects/highlights

```

```

891                                     // the button.
892
893     } else if (event.getSource() == currentInstance.button[0]) // This means
894                                     // that for
895         || event.getSource() == currentInstance.button[1] // any button
896                                     // on any
897         || event.getSource() == currentInstance.button[2] // cursor will
898                                     // change to
899                                     // hand
900
901         || event.getSource() == currentInstance.button[3]) {
902         this.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
903
904     } else if (event.getSource() == awInstance.inputField // This does not
905         || event.getSource() == bd1Instance.inputField // change the
906                                     // cursor.
907         || event.getSource() == bd2Instance.inputField) // but removes
908                                     // the red
909         currentInstance.inputField.setBackground(Color.GREEN); // background
910                                     // (red
911                                     // background
912                                     // is added,
913                                     // when user
914                                     // enters
915                                     // wrong
916                                     // answer.
917         refresh();
918     }
919 }
920
921 @Override
922 public void mouseDragged(MouseEvent e) /**/// Following methods are not
923                                     // used,
924     // but have to be present, because
925     // of implementation of mouse and
926     // action listeners.
927     // TODO Auto-generated method stub
928 }
929
930
931 @Override
932 public void mouseExited(MouseEvent e) /**/// Not used
933     // TODO Auto-generated method stub
934 }
935
936
937 @Override
938 public void mousePressed(MouseEvent e) /**/// Not used
939     // TODO Auto-generated method stub
940 }
941
942
943 @Override
944 public void mouseReleased(MouseEvent e) /**/// Not used
945     // TODO Auto-generated method stub
946 }
947
948
949 @Override
950 public void mouseEntered(MouseEvent e) /**/// Not used
951     // TODO Auto-generated method stub
952 }
953 }
954

```



MainWindow.java

955

956