Bereich: Graphische Benutzeroberflächen (UI), Events (2)

Editor	**Musterlösung**
Package: `de.dhbwka.java.exercise.ui.editor`	**Klasse:** `Editor`

```
package de.dhbwka.java.exercise.ui.editor;

import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JTextPane;
import javax.swing.KeyStroke;
import javax.swing.filechooser.FileFilter;

/**
 *
 * Part of lectures on 'Programming in Java'. Baden-Wuerttemberg
 * Cooperative State University.
 *
 * (C) 2016-2018 by W. Geiger, T. Schlachter, C. Schmitt, W. Suess
 *
 * @author DHBW lecturer
 * @version 1.4
 */
@SuppressWarnings("serial")
public class Editor extends JFrame {
   private static final String SEPARATOR = "--";
   private static final String SEND_MENU = "-SENDMENU-";
   private static final String NEWFILENAME = "newfile.txt";

   private final static int INITIAL_WIDTH = 600;
   private final static int INITIAL_HEIGHT = 480;

   private JMenu fileMenu;
   private JMenu editMenu;
   private JTextPane editPane;

   private String filePath = Editor.NEWFILENAME;
```

```
private final String[] fileMenuItems = { "Neu", "\u00D6ffnen",
      Editor.SEPARATOR, "Schlie\u00DFen", Editor.SEPARATOR, "Speichern",
      "Speichern unter...", "Als Webseite speichern",
      "Suchen", Editor.SEPARATOR, "Versionen", Editor.SEPARATOR,
      "Webseitenvorschau", Editor.SEPARATOR, "Seite einrichten...",
      "Seitenansicht", "Drucken", Editor.SEPARATOR, Editor.SEND_MENU,
      "Eigenschaften", Editor.SEPARATOR, "bilanz_2017.doc",
      "bericht_2018_01.doc", "ziele.doc", Editor.SEPARATOR,
      "Beenden" };

// The following numbers are the array indices of the items in fileMenuItems
private static final int NEW_MENU = 0;
private static final int OPEN_MENU = 1;
private static final int CLOSE_MENU = 3;
private static final int SAVE_MENU = 5;
private static final int SAVEAS_MENU = 6;
private static final int EXIT_MENU = 25;

private final String[] editMenuItems = { "R\u00FCckg\u00E4ngig",
      "Wiederholen", Editor.SEPARATOR, "Ausschneiden", "Kopieren",
      "Office-Zwischenablage", "Einf\u00FCgen", "Inhalte einf\u00FCgen",
      "Als Hyperlink einf\u00FCgen", Editor.SEPARATOR, "L\u00F6schen",
      "Alles markieren", Editor.SEPARATOR, "Suchen...", "Ersetzen...",
      "Gehe zu...", Editor.SEPARATOR, "Verkn\u00FCpfungen...", "Objekt" };

// The following numbers are the array indices of the items in editMenuItems
private static final int FIND_MENU = 13;
private static final int REPLACE_MENU = 14;

private final String[] sendMenuItems = { "E-Mail-Empf\u00E4nger",
      "E-Mail-Empf\u00E4nger (zur \u00DCberarbeitung)",
      "E-Mail-Empf\u00E4nger (als Anlage)", Editor.SEPARATOR,
      "Verteilerempf\u00E4nger...", "Onlinebesprechungsteilnehmer",
      "Exchange-Ordner...", "Fax-Empf\u00E4nger...", Editor.SEPARATOR,
      "Microsoft Powerpoint" };

/**
 * Constructor for the editor
 */
public Editor() {
   super( "Editor" );
   this.setLayout( new BorderLayout() );

   // Create scrollable editor
   this.editPane = new JTextPane();
   this.editPane.setFont( new Font( Font.MONOSPACED, Font.PLAIN, 16 ) );
   this.add( new JScrollPane( this.editPane ), BorderLayout.CENTER );

   // Create menu
   JMenuBar menu = new JMenuBar();
   menu.add( this.fileMenu =
         this.getMenu( "Datei", KeyEvent.VK_D, this.fileMenuItems ) );
   menu.add( this.editMenu =
         this.getMenu( "Bearbeiten", KeyEvent.VK_B, this.editMenuItems ) );

   // Add listeners
```

```
      this.addFileMenuListeners();
      this.addEditMenuListeners(); // extension

      this.setSaveAndCloseEnabledState();
      this.setJMenuBar( menu );
      this.setSize( Editor.INITIAL_WIDTH, Editor.INITIAL_HEIGHT );
      this.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      this.setVisible( true );
   }

   /**
    * Add the listeners to the file menu items
    */
   private void addFileMenuListeners() {
      // Cast is not very elegant -> use getItem() !
      JMenuItem itemNew =
            (JMenuItem) this.fileMenu.getMenuComponent( Editor.NEW_MENU );
      itemNew.addActionListener( new ActionListener() {
         @Override
         public void actionPerformed( ActionEvent e ) {
            Editor.this.filePath = Editor.NEWFILENAME;
            Editor.this.setTitle( Editor.this.filePath );
            Editor.this.editPane.setText( "" );
            Editor.this.setSaveAndCloseEnabledState();
         }
      } );

      // Open
      this.fileMenu.getItem( Editor.OPEN_MENU )
            .addActionListener( new ActionListener() {
               @Override
               public void actionPerformed( ActionEvent e ) {
                  File f = Editor.this.selectFile( "\u00D6ffnen" );
                  if ( f != null ) {
                     Editor.this.filePath = f.getAbsolutePath();
                     Editor.this.setTitle( f.getName() );
                     Editor.this.editPane
                           .setText( Editor.this.readTextfile( f ) );
                     Editor.this.setSaveAndCloseEnabledState();
                  }
               }
            } );

      // Close (extension)
      this.fileMenu.getItem( Editor.CLOSE_MENU )
            .addActionListener( new ActionListener() {
               @Override
               public void actionPerformed( ActionEvent e ) {
                  Editor.this.filePath = Editor.NEWFILENAME;
                  Editor.this.setTitle( Editor.this.filePath );
                  Editor.this.editPane.setText( "" );
                  Editor.this.setSaveAndCloseEnabledState();
               }
            } );


      // Save
```

```
      this.fileMenu.getItem( Editor.SAVE_MENU )
            .addActionListener( new ActionListener() {
               @Override
               public void actionPerformed( ActionEvent e ) {
                  Editor.this.writeTextfile( new File( Editor.this.filePath ),
                        Editor.this.editPane.getText() );
               }
            } );

      // Save as (extension)
      this.fileMenu.getItem( Editor.SAVEAS_MENU )
            .addActionListener( new ActionListener() {
               @Override
               public void actionPerformed( ActionEvent e ) {
                  File f = Editor.this.selectFile( "Speichern unter" );
                  if ( f != null ) {
                     Editor.this.filePath = f.getAbsolutePath();
                     Editor.this.setTitle( f.getName() );
                     Editor.this.writeTextfile( f,
                           Editor.this.editPane.getText() );
                     Editor.this.setSaveAndCloseEnabledState();
                  }
               }
            } );

      // Exit (with confirmation)
      this.fileMenu.getItem( Editor.EXIT_MENU )
            .addActionListener( new ActionListener() {
               @Override
               public void actionPerformed( ActionEvent e ) {
                  if ( JOptionPane.showConfirmDialog( null,
                        "Editor wirklich beenden?",
                        "Beenden",
                        JOptionPane.YES_NO_OPTION ) == JOptionPane.YES_OPTION ) {
                     System.exit( 0 );
                  }
               }
            } );
   }
   /**
    * Add the listeners to the edit menu items (extension)
    */
   private void addEditMenuListeners() {
      JMenuItem itemFind = this.editMenu.getItem( Editor.FIND_MENU );
      itemFind.setAccelerator( KeyStroke.getKeyStroke( "control F" ) );

      itemFind.addActionListener( new ActionListener() {
         @Override
         public void actionPerformed( ActionEvent e ) {
            String toFind =
                  JOptionPane.showInputDialog( null, "Was suchen?",
                        Editor.this.editPane.getSelectedText() );

            // Force unix line separators
            String text = Editor.this.editPane.getText()
                  .replaceAll( System.lineSeparator(), "\n" );
            // If something is already selected: continue search afterwards
```

```
            int start = Editor.this.editPane.getSelectionStart();
            start = (start >= 0 && start < text.length()) ? start + 1 : 0;
            int position = text.indexOf( toFind, start );
            if ( position >= 0 ) {
               Editor.this.editPane.select( position,
                     position + toFind.length() );
            }
         }
      } );
      JMenuItem itemReplace = this.editMenu.getItem( Editor.REPLACE_MENU );
      itemReplace.setAccelerator( KeyStroke.getKeyStroke( "control G" ) );
      itemReplace.addActionListener( new ActionListener() {
         @Override
         public void actionPerformed( ActionEvent e ) {
            String toReplace =
                  JOptionPane.showInputDialog( null, "Was suchen?",
                        Editor.this.editPane.getSelectedText() );
            String replacement =
                  JOptionPane.showInputDialog( null, "Mit was ersetzen?",
                        Editor.this.editPane.getSelectedText() );

            // Force unix line separators
            String text = Editor.this.editPane.getText()
                  .replaceAll( System.lineSeparator(), "\n" );

            text = text.replaceAll( toReplace, replacement );
            Editor.this.editPane.setText( text );
         }
      } );
   }
   /**
    * Produces a JMenu with JMenuItems according to items array
    *
    * @param menuName name of parent menu
    * @param mnemonic mnemonic for the parent menu
    * @param items string array with sub items labels
    * @return create parent menu
    */
   private JMenu getMenu( String menuName, int mnemonic, String[] items ) {
      JMenu menu = new JMenu( menuName );
      menu.setMnemonic( mnemonic );
      for ( String menuItemName : items ) {
         switch (menuItemName) {
            case SEPARATOR:
               menu.add( new JSeparator() );
               break;
            case SEND_MENU:
               menu.add( this.getMenu( "Senden an", KeyEvent.VK_S,
                     this.sendMenuItems ) );
               break;
            default:
               menu.add( new JMenuItem( menuItemName ) );
         }
      }
      return menu;
   }
   /**
```

```
    * Set the menu items enabled state according to condition if current file
    * path is {@link Editor#NEWFILENAME}.
    */
   private void setSaveAndCloseEnabledState() {
      boolean enabled = !this.filePath.equals( Editor.NEWFILENAME );
      this.fileMenu.getItem( Editor.SAVE_MENU ).setEnabled( enabled );
      this.fileMenu.getItem( Editor.CLOSE_MENU ).setEnabled( enabled );
   }

   /**
    * Open file select dialog
    *
    * @param text approve button label
    * @return selected file or <code>null</code> if no file was selected
    */
   public File selectFile( String text ) {
      JFileChooser fc = new JFileChooser();
      fc.setFileFilter( new FileFilter() {
         @Override
         public boolean accept( File f ) {
            return f.isDirectory()
                  || f.getName().toLowerCase().endsWith( ".txt" );
         }

         @Override
         public String getDescription() {
            return "Textdateien";
         }
      } );
      if ( fc.showDialog( null, text ) == JFileChooser.APPROVE_OPTION ) {
         return fc.getSelectedFile();
      }
      return null;
   }

   /**
    * Get the content of a text file as string and current system line
    * separators
    *
    * @param textFile text file to read
    * @return content of file or empty string if file does not exist
    */
   protected String readTextfile( File textFile ) {
      StringBuilder result = new StringBuilder();
      if ( textFile.exists() ) {
         try ( BufferedReader br =
               new BufferedReader( new FileReader( textFile ) ) ) {
            while ( br.ready() ) {
               result.append( br.readLine() ).append( System.lineSeparator() );
            }
         } catch ( IOException e ) {
            System.err.println( "Fehler beim Lesen: " + e.getMessage() );
         }
      }
      return result.toString();
   }
   /**
```

```
    * Save content to text file (overwriting existing files!)
    *
    * @param textFile
    *           file to write to
    * @param content
    *           content to write
    */
   public void writeTextfile( File textFile, String content ) {
      if ( textFile != null ) {
         try ( BufferedWriter bw =
               new BufferedWriter( new FileWriter( textFile ) ) ) {
            bw.write( content );
         } catch ( IOException ex ) {
            System.err.println( "Fehler beim Schreiben von  " +
                  textFile.getAbsolutePath() );
         }
      }
   }

   /**
    * Run an instance of the editor
    */
   public static void main( String[] args ) {
      new Editor();
   }

}
```

Bereich: Graphische Benutzeroberflächen (UI), Events (2)

Hütchenspiel	**Musterlösung**
Package: `de.dhbwka.java.exercise.ui.event`	**Klasse:** `ShellGame`

```java
package de.dhbwka.java.exercise.ui.event;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

/**
 * Part of lectures on 'Programming in Java'. Baden-Wuerttemberg
 * Cooperative State University.
 *
 * (C) 2016-2018 by W. Geiger, T. Schlachter, C. Schmitt, W. Suess
 *
 * @author DHBW lecturer
 * @version 1.1
 */
@SuppressWarnings( "serial" )
public class ShellGame extends JFrame {

   private final static int LIMIT = 3; // Number of Buttons
   private final static String STATS_FILENAME = "shellstats.txt";

   private final Random rnd = new Random();

   private int shell;
   private int attempts = 0;
   private JButton[] shellButtons;
   private JButton btnExit = new JButton( "Exit" );
   private JButton btnStat = new JButton( "Statistics" );
   private JTextField txtName = new JTextField( "Name", 20 );
   private JTextField txtOutput = new JTextField( 30 );



   // Continued on next page
```

```
/**
 * Create shell game
 */
public ShellGame() {
   super( "Shell Game" );
   this.setLayout( new GridLayout( 4, 1 ) );

   JPanel panName = new JPanel();
   JPanel panShell = new JPanel();
   JPanel panFunction = new JPanel();
   JPanel panOutput = new JPanel();

   panName.add( new JLabel( "Player Name" ) );
   panName.add( this.txtName );

   this.shellButtons = new JButton[ShellGame.LIMIT];
   for ( int i = 0; i < ShellGame.LIMIT; i++ ) {
      JButton btnShell = new JButton( "Shell " + (i + 1) );
      btnShell.setActionCommand( Integer.toString( i + 1 ) );
      this.shellButtons[i] = btnShell;
      panShell.add( btnShell );
   }

   panOutput.add( this.btnStat );
   panOutput.add( this.btnExit );

   panFunction.add( this.txtOutput );

   this.add( panName );
   this.add( panShell );
   this.add( panOutput );
   this.add( panFunction );

   this.initEventHandling();

   this.shell = this.getRandomShell();

   this.setSize( 400, 300 );
   this.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   this.setVisible( true );
}

/**
 * Add event handling to buttons
 */
public void initEventHandling() {
   // Statistics
   this.btnStat.addActionListener( new ActionListener() {
      @Override
      public void actionPerformed( ActionEvent event ) {
         ShellGame.this.showStats();
      }
   } );

   // Continued on next page
```

```
      // Exit
      this.btnExit.addActionListener( new ActionListener() {
         @Override
         public void actionPerformed( ActionEvent event ) {
            System.exit( 0 );
         }
      } );

      // Shell-Listener for all Shell Buttons
      ActionListener shellListener = new ActionListener() {
         @Override
         public void actionPerformed( ActionEvent event ) {
            // Get Shell Number by ActionCommand
            ShellGame.this
                  .revealShell( Integer.parseInt( event.getActionCommand() ) );
         }
      };
      for ( JButton b : this.shellButtons ) {
         b.addActionListener( shellListener );
      }
   }

   /**
    * Reveal a shell
    */
   public void revealShell( int tip ) {
      this.attempts++;
      if ( tip == this.shell ) {
         this.txtOutput.setText( "Attempt " + this.attempts
               + " wins: Ball was below shell " + this.shell );
         this.saveStats();
         this.attempts = 0;
      } else {
         this.txtOutput.setText( "Attempt " + this.attempts
               + " fails: Ball was below shell " + this.shell );
      }
      this.shell = this.getRandomShell();
   }
   /**
    * Show statistics from stats file
    */
   private void showStats() {
      try ( BufferedReader in =
            new BufferedReader( new FileReader( ShellGame.STATS_FILENAME ) ) ) {
         int count = 0;
         float sum = 0;
         while ( in.ready() ) {
            try {
               sum += Integer.parseInt( in.readLine().split( ";" )[1] );
               count++;
            } catch ( Exception e ) {
            }
         }
         this.txtOutput.setText( "Success after " + sum / count + " attempts!" );
      } catch ( IOException eee ) {
      }
   }
```

```
   /**
    * Add statistics to stats file
    */
   private void saveStats() {
      try ( PrintWriter f = new PrintWriter(
            new FileWriter( ShellGame.STATS_FILENAME, true ) ) ) {
         f.println( this.txtName.getText() + ";" + this.attempts );
      } catch ( IOException e ) {
      }
   }

   /**
    * Gets a random number for shell from 1..LIMIT
    */
   private int getRandomShell() {
      return this.rnd.nextInt( ShellGame.LIMIT ) + 1;
   }

   public static void main( String args[] ) {
      new ShellGame();
   }
}
```