Komunikator

application source code

```java
import pl.nalazek.komunikator.*;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

public class Komunikator {

        public static void main(String[] a)
        {
            new Startuj();
        }
}

```

```java
package pl.nalazek.komunikator;

import pl.nalazek.komunikator.gui.Gui;
import pl.nalazek.komunikator.logika.Model;
import pl.nalazek.komunikator.logika.Sterowanie;
import pl.nalazek.komunikator.logika.Watek;
import pl.nalazek.komunikator.logika.WatekSluchajacy;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa typu Singleton, przechowująca referencję trzech podstawowych komponentów programu (MVC) */
public class Program {

    private static Model model ;
    private static Gui gui;
    private static Sterowanie sterowanie;
    private static Program referencja;

    protected Program()
    {}

    /** Konstruktor klasy
     * @param gui Referencja do interfejsu graficznego w postaci klasy Gui
     * @param model Referencja do modelu programu w postaci klasy Model
     * @param sterowanie Referencja do sterowania programu w postaci klasy Sterowanie
     * @return Referencja do programu w postaci klasy program
     */
    public static Program utworz(Gui gui, Model model, Sterowanie sterowanie)
    {
        if(referencja == null)
        {
            referencja = new Program();
        }
        Program.model = model;
        Program.gui = gui;
        Program.sterowanie = sterowanie;
        return referencja;
    }

    /** Metoda umożliwiająca dostęp do metod zawartych w modelu programu,
     * dostępna tylko dla unikalnego egzemplarza klasy Sterowanie, przekazanego wcześniej w postaci referencji
     * w konstruktorze klasy Program.
     * @param s Referencja do sterowania programu
     * @return Referencja do modelu programu lub "null" w przypadku niepoprawnej weryfikacji
     */
    public static Model komponentModel(Sterowanie s)
    {
        if(s!=null && s == sterowanie)
        return model;
        else return null;
    }

    /** Metoda umożliwiająca dostęp do metod zawartych w interfejsie graficznym programu,
     * dostępna tylko dla unikalnego egzemplarza klasy Sterowanie, przekazanego wcześniej
     * w postaci referencji w konstruktorze klasy Program.
     * @param s Referencja do sterowania programu
     * @return Referencja do interfejsu graficznego programu lub "null" w przypadku niepoprawnej weryfikacji
     */
    public static Gui komponentGui(Sterowanie s)
    {
        if(s!=null && s == sterowanie)
        return gui;
        else return null;
    }

    /** Metoda umożliwiająca dostęp do metod zawartych w interfejsie graficznym programu,
     * dostępna tylko dla wątków słuchających w programie
     * @param s Referencja do klasy WatekSluchajacy
     * @return Referencja do interfejsu graficznego programu lub "null" w przypadku niepoprawnej weryfikacji
     */
    public static Gui komponentGui(WatekSluchajacy s)
    {
        if(s!=null && (Watek.sprawdzCzyIstniejeWatek(s)))
        return gui;
        else return null;
```

```
79      }
80
81      /** Metoda umożliwiająca dostęp do metod zawartych w sterowaniu, dostępna dla Modelu i Gui
82       * @return Referencja do sterowania
83       */
84      public static Sterowanie komponentSterowanie()
85      {
86          return sterowanie;
87      }
88
89      /** Metoda uruchamiająca sterowanie programu
90       * Musi zostać uruchomiona po wywołaniu konstruktora */
91      public void uruchom()
92      {
93          sterowanie.sterowanieStart();
94      }
95
96
97  }
98
```

```java
package pl.nalazek.komunikator;

import pl.nalazek.komunikator.gui.Gui;
import pl.nalazek.komunikator.logika.Model;
import pl.nalazek.komunikator.logika.Sterowanie;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Główna klasa konstruująca program */
public class Startuj {

    /** Konstruktor klasy */
    public Startuj()
    {
    Program.utworz(new Gui(),new Model(),new Sterowanie()).uruchom();
    }
}

```

```java
package pl.nalazek.komunikator.gui;


import java.awt.*;
import javax.swing.*;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca interfejs graficzny programu */
public class Gui extends JFrame {

    /** Napis nad polem z romową */
    public JLabel       panel1Etykieta;
    /** Pole tekstowe z rozmową */
    public JTextPane    panel1PoleTekstowesc;
    /** Pole tekstowe do wpisywania wiadomości */
    public JTextArea    panel1PoleTekstoweWpissc;
    /** Przycisk "Czyść" */
    public JButton      panel1Wymaz;
    /** Przycisk "Wyślij" */
    public JButton      panel1Wyslij;
    /** Przycisk "Wyślij plik" */
    public JButton      panel1WyslijPlik;
    /** Przycisk "Połącz" */
    public JButton      panel2Polacz;
    /** Przycisk "Rozłącz" */
    public JButton      panel2Rozlacz;
    /** Przycisk "Dodaj" */
    public JButton      panel2Dodaj;
    /** Przycisk "Usuń" */
    public JButton      panel2Usun;
    /** Kratka do zaznaczania online */
    public JCheckBoxMenuItem menuOnline;
    /** Napis na pasku stanu */
    public JLabel       napisPaska;
    /** Lista wyświetlanych użytkowników */
    public JList<String>    listaUzytkownikow;
    /** Pasek postępu */
    public JProgressBar pasekPostepu;
    /** Wskaźnik zaznaczonego elementu na liście użytkowników */
    public int listaOstatniIndeks = -1;
    /** Pole wyboru "Internet" */
    JRadioButtonMenuItem onlnInternet;
    /** Pole wyboru "Localhost1" */
    JRadioButtonMenuItem onlnLchst1;
    /** Pole wyboru "Localhost2" */
    JRadioButtonMenuItem onlnLchst2;
    /** Wskazuje zaznaczone pole wyboru */
    int menuOnlineWybor = 1;

    private OknoPanelRozmowa panelRozmowa;
    private OknoPanelUzytkownicy panelUzytkownicy;
    private OknoMenu oknoMenu;
    private static final long serialVersionUID = 1L;

    /** Konstruktor interfejsu graficznego */
        public Gui()
        {
            // Utworzenie zarządców obsługujących zdarzenia przychodzące z okna programu
            panelRozmowa = new OknoPanelRozmowa(this);
            panelUzytkownicy = new OknoPanelUzytkownicy(this);
            oknoMenu = new OknoMenu(this);
            addWindowListener(new Okno(this));

            SwingUtilities.invokeLater(new Runnable()
            {
                public void run() {
                    getRootPane().setWindowDecorationStyle(JRootPane.FRAME);
                    setUndecorated(true);


                    // Budowa panelu rozmowy
                    JPanel      panel1 = new JPanel();
                    GroupLayout layout = new GroupLayout(panel1);
                                panel1.setLayout(layout);
```

```
                    layout.setAutoCreateContainerGaps(true);
                    layout.setAutoCreateGaps(true);
                    panel1Etykieta = new JLabel("Nie połączono");

                    panel1PoleTekstowesc = new JTextPane();
                        panel1PoleTekstowesc.setEditable(false);
                JScrollPane panel1PoleTekstowe = new JScrollPane(panel1PoleTekstowesc);
                        panel1PoleTekstowe.setAutoscrolls(true);
                        panel1PoleTekstowe.setMinimumSize(new Dimension(300,200));
                    panel1PoleTekstoweWpissc = new JTextArea("Wpisz wiadomość");
                        panel1PoleTekstoweWpissc.setLineWrap(true);
                        panel1PoleTekstoweWpissc.setAutoscrolls(true);
                JScrollPane panel1PoleTekstoweWpis = new JScrollPane(panel1PoleTekstoweWpissc);
                        panel1PoleTekstoweWpis.setMinimumSize(new Dimension(300,42));

                        panel1Wymaz = new JButton("Czy\u015b\u0107");
                        panel1Wymaz.addActionListener(panelRozmowa);
                    panel1Wyslij = new JButton("Wy\u015blij");
                        panel1Wyslij.addActionListener(panelRozmowa);
                    panel1WyslijPlik = new JButton("Wy\u015blij plik");
                        panel1WyslijPlik.addActionListener(panelRozmowa);

                layout.setHorizontalGroup(layout.createParallelGroup()
                        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                                .addComponent(panel1Etykieta)
                                .addComponent(panel1PoleTekstowe)
                                .addComponent(panel1PoleTekstoweWpis))
                        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                                .addGroup(layout.createSequentialGroup()
                                        .addComponent(panel1Wyslij)
                                        .addComponent(panel1Wymaz))
                                .addComponent(panel1WyslijPlik)));

                layout.setVerticalGroup(layout.createSequentialGroup()
                        .addComponent(panel1Etykieta)
                        .addComponent(panel1PoleTekstowe)
                        .addComponent(panel1PoleTekstoweWpis)
                        .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                                .addComponent(panel1Wymaz)
                                .addComponent(panel1Wyslij))
                        .addComponent(panel1WyslijPlik));


                //Budowa panelu paska informacyjnego
                JPanel pasek = new JPanel();
                napisPaska = new JLabel("Offline");
                napisPaska.setAlignmentX(Component.RIGHT_ALIGNMENT);
                pasek.setMinimumSize(new Dimension(150,10));
                pasek.setLayout(new BoxLayout(pasek,BoxLayout.Y_AXIS));
                pasek.add(napisPaska);

                //Budowa panelu paska postępu
                pasekPostepu = new JProgressBar();
                pasekPostepu.setIndeterminate(true);
                pasekPostepu.setVisible(false);

                //Budowa panelu użytkowników
                JPanel panel2 = new JPanel();
                JLabel      panel2Tytul = new JLabel("Użytkownicy:");
                        panel2Polacz = new JButton("Po\u0142\u0105cz");
                            panel2Polacz.addActionListener(panelUzytkownicy);
                        panel2Rozlacz = new JButton("Roz\u0142\u0105cz");
                            panel2Rozlacz.addActionListener(panelUzytkownicy);
                        panel2Dodaj = new JButton("Dodaj");
                            panel2Dodaj.addActionListener(panelUzytkownicy);
                        panel2Usun = new JButton("Usu\u0144");
                            panel2Usun.addActionListener(panelUzytkownicy);
                JList<String>   panel2Listaa = new JList<String>();
                                panel2Listaa.setVisibleRowCount(10);
                                listaUzytkownikow = panel2Listaa;
                                listaUzytkownikow.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                                listaUzytkownikow.addListSelectionListener(panelUzytkownicy);
                JScrollPane panel2Lista = new JScrollPane(panel2Listaa);
                        panel2Lista.setMinimumSize(new Dimension(158,220));
                        panel2Lista.setMaximumSize(new Dimension(158,1000));

                GroupLayout layout2 = new GroupLayout(panel2);
                panel2.setLayout(layout2);
```

```
157                     layout2.setAutoCreateContainerGaps(true);
158                     layout2.setAutoCreateGaps(true);
159 
160                     layout2.setHorizontalGroup(layout2.createParallelGroup()
161                             .addGroup(layout2.createParallelGroup(GroupLayout.Alignment.LEADING)
162                                 .addComponent(panel2Tytul)
163                                 .addComponent(panel2Lista))
164                             .addGroup(layout2.createParallelGroup(GroupLayout.Alignment.TRAILING)
165                                     .addGroup(layout2.createSequentialGroup()
166                                         .addComponent(panel2Polacz)
167                                         .addComponent(panel2Rozlacz))
168                                     .addGroup(layout2.createSequentialGroup()
169                                         .addComponent(panel2Dodaj)
170                                         .addComponent(panel2Usun)))
171                             );
172 
173                     layout2.setVerticalGroup(layout2.createSequentialGroup()
174                             .addComponent(panel2Tytul)
175                             .addComponent(panel2Lista)
176                             .addGroup(layout2.createParallelGroup(GroupLayout.Alignment.BASELINE)
177                                     .addComponent(panel2Polacz)
178                                     .addComponent(panel2Rozlacz))
179                             .addGroup(layout2.createParallelGroup(GroupLayout.Alignment.BASELINE)
180                                     .addComponent(panel2Dodaj)
181                                     .addComponent(panel2Usun))
182                                     );
183 
184                     //Budowa menu programu
185                     JMenuBar menu = new JMenuBar();
186                     setJMenuBar(menu);
187                     JMenu plik = new JMenu("Plik");
188                     menu.add(plik);
189                         JMenuItem zakoncz = new JMenuItem("Zako\u0144cz");
190                                 zakoncz.addActionListener(oknoMenu);
191                         plik.add(zakoncz);
192                     JMenu ustawienia = new JMenu("Ustawienia");
193                     menu.add(ustawienia);
194                         ustawienia.add(menuOnline = new JCheckBoxMenuItem("Online"));
195                             menuOnline.addActionListener(oknoMenu);
196                         ustawienia.addSeparator();
197                         ButtonGroup wyborOnline = new ButtonGroup();
198                             onlnInternet = new JRadioButtonMenuItem("Internet");
199                             onlnLchst1 = new JRadioButtonMenuItem("Localhost 1");
200                             onlnLchst2 = new JRadioButtonMenuItem("Localhost 2");
201                             onlnInternet.addItemListener(oknoMenu);
202                             onlnLchst1.addItemListener(oknoMenu);
203                             onlnLchst2.addItemListener(oknoMenu);
204                             wyborOnline.add(onlnInternet);
205                             wyborOnline.add(onlnLchst1);
206                             wyborOnline.add(onlnLchst2);
207                                 onlnLchst1.setSelected(true);
208                         ustawienia.add(onlnInternet);
209                         ustawienia.add(onlnLchst1);
210                         ustawienia.add(onlnLchst2);
211                     JMenu pomoc = new JMenu("Pomoc");
212                     menu.add(pomoc);
213                         JMenuItem oProgramie = new JMenuItem("O programie...");
214                                     oProgramie.addActionListener(oknoMenu);
215                         pomoc.add(oProgramie);
216 
217 
218 
219 
220                     //Budowa głównego okna
221                     GroupLayout layout0 = new GroupLayout(getContentPane());
222                     getContentPane().setLayout(layout0);
223                     layout0.setAutoCreateContainerGaps(true);
224                     layout0.setAutoCreateGaps(true);
225 
226                     layout0.setHorizontalGroup(layout0.createParallelGroup(GroupLayout.Alignment.TRAILING)
227                             .addGroup(layout0.createSequentialGroup()
228                                     .addComponent(panel1)
229                                     .addComponent(panel2))
230                             .addGroup(layout0.createSequentialGroup()
231                                     .addComponent(pasekPostepu)
232                                     .addComponent(pasek)) );
233 
234                     layout0.setVerticalGroup(layout0.createSequentialGroup()
```

```
235                             .addGroup(layout0.createParallelGroup(GroupLayout.Alignment.BASELINE)
236                                     .addComponent(panel1)
237                                     .addComponent(panel2))
238                             .addGroup(layout0.createParallelGroup(GroupLayout.Alignment.BASELINE)
239                                     .addComponent(pasekPostepu)
240                                     .addComponent(pasek)) );
241
242                 pack();
243                 setVisible(true);
244             }
245         } );
246
247     }
248
249     /** Dezaktywuje przyciski panelu rozmowa */
250     public void dezaktywujPrzyciskiPaneluRozmowa()
251     {
252         SwingUtilities.invokeLater(new Runnable()
253         {
254             public void run() {
255         panel1Wymaz.setEnabled(false);
256         panel1Wyslij.setEnabled(false);
257         panel1WyslijPlik.setEnabled(false);
258         }
259         } );
260     }
261
262     /** Dezaktywuje cały panel rozmowa */
263     public void dezaktywujPanelRozmowa()
264     {
265         SwingUtilities.invokeLater(new Runnable()
266         {
267             public void run() {
268         dezaktywujPrzyciskiPaneluRozmowa();
269         panel1PoleTekstoweWpissc.setEnabled(false);
270             }
271         } );
272     }
273
274     /** Aktywuje przyciski panelu rozmowa */
275     public void aktywujPrzyciskiPaneluRozmowa()
276     {
277         SwingUtilities.invokeLater(new Runnable()
278         {
279             public void run() {
280         panel1Wymaz.setEnabled(true);
281         panel1Wyslij.setEnabled(true);
282         panel1WyslijPlik.setEnabled(true);
283             }
284         } );
285     }
286
287     /** Aktywuje cały panel rozmowa */
288     public void aktywujPanelRozmowa()
289     {
290         SwingUtilities.invokeLater(new Runnable()
291         {
292             public void run() {
293         aktywujPrzyciskiPaneluRozmowa();
294         panel1PoleTekstoweWpissc.setEnabled(true);
295             }
296         } );
297     }
298
299     /** Dezaktywuje przyciski panelu użytkownicy */
300     public void dezaktywujPrzyciskiPaneluUzytkownicy()
301     {
302         dezaktywujPrzyciskiPaneluUzytkownicyLaczenie();
303         SwingUtilities.invokeLater(new Runnable()
304         {
305             public void run() {
306
307         panel2Dodaj.setEnabled(false);
308         panel2Usun.setEnabled(false);
309             }
310         } );
311     }
312
```

```
313         /** Aktywuje przyciski panelu użytkownicy */
314         public void aktywujPrzyciskiPaneluUzytkownicy()
315         {
316             SwingUtilities.invokeLater(new Runnable()
317             {
318                 public void run() {
319 
320             panel2Dodaj.setEnabled(true);
321             panel2Usun.setEnabled(true);
322                 }
323             } );
324         }
325 
326         /** Dezktywuje przyciski panelu użytkownicy związanie z połączeniami */
327         public void dezaktywujPrzyciskiPaneluUzytkownicyLaczenie()
328         {
329             SwingUtilities.invokeLater(new Runnable()
330             {
331                 public void run() {
332             panel2Polacz.setEnabled(false);
333             panel2Rozlacz.setEnabled(false);
334                 }
335             } );
336         }
337 
338         /** Aktywuje przyciski panelu użytkownicy związanie z połączeniami */
339         public void aktywujPrzyciskiPaneluUzytkownicyLaczenie()
340         {
341             SwingUtilities.invokeLater(new Runnable()
342             {
343                 public void run() {
344             panel2Polacz.setEnabled(true);
345             panel2Rozlacz.setEnabled(true);
346                 }
347             } );
348         }
349 
350         /** Wyświetla okno dialogowe typu "Ostrzeżenie" z podaną w argumencie treścią *
351          * @param s Treść ostrzeżenia
352          */
353         public void oknoDialogoweWyjatek(String s)
354         {
355             JOptionPane.showMessageDialog(this, s, "Ostrze\u017cenie", JOptionPane.WARNING_MESSAGE);
356         }
357 
358         /** Wyświetla okno dialogowe typu "Pytanie" z podaną w argumencie treścią *
359          * @param s Treść pytanie
360          * @return Liczbę odpowiadającą wyborowi użytkownika (1:tak, 0:nie, -1:anuluj)
361          */
362         public int oknoDialogowePytanie(String s)
363         {
364             int wynik = JOptionPane.showConfirmDialog(this, s, "Pytanie", JOptionPane.YES_NO_CANCEL_OPTION);
365             if(wynik == JOptionPane.YES_OPTION) return 1;
366             else if(wynik == JOptionPane.NO_OPTION) return 0;
367             else return -1;
368         }
369 
370         /** Dezaktywuje możliwość zmieniania wyboru trybu w jakim program pracuje online */
371         public void dezaktywujWyborPrzelacznikowOnline()
372         {
373             SwingUtilities.invokeLater(new Runnable()
374             {
375             public void run() {
376             onlnInternet.setEnabled(false);
377             onlnLchst1.setEnabled(false);
378             onlnLchst2.setEnabled(false);
379             }}) ;
380         }
381 
382         /** Wyświetla okno dialogowe z informacjami na temat programu */
383         public void oknoDialogoweInfo()
384         {
385             JOptionPane.showMessageDialog(this, "Komunikator\n"
386                     + "\n"
387                     + "Wersja: 1.0\n"
388                     + "Autor: Daniel Nalazek\n"
389                     + "","O programie", JOptionPane.INFORMATION_MESSAGE);
390         }
```

```
391
392
393         /** Wyświetla okno dialogowe typu "Informacja" z podaną w argumentach treścią *
394          * @param tekst Treść informacji
395          * @param tytul Tytuł okna
396          */
397         public void oknoDialogoweInfo(String tekst, String tytul)
398         {
399             JOptionPane.showMessageDialog(this, tekst ,tytul, JOptionPane.INFORMATION_MESSAGE);
400         }
401
402         /** Aktywuje możliwość zmieniania wyboru trybu w jakim program pracuje online */
403         public void aktywujWyborPrzelacznikowOnline()
404         {
405             SwingUtilities.invokeLater(new Runnable()
406             {
407             public void run() {
408             onlnInternet.setEnabled(true);
409             onlnLchst1.setEnabled(true);
410             onlnLchst2.setEnabled(true);
411             }}) ;
412         }
413
414 }
415
```

```java
package pl.nalazek.komunikator.gui;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import pl.nalazek.komunikator.Program;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca zarządce obsługi całego okna programu */
public class Okno extends WindowAdapter{

        private Gui gui;

        /** Konstruktor
         * @param gui Referencja do interfejsu graficznego programu */
        public Okno(Gui gui)
        {
            this.gui = gui;
        }

        /** Funkcja odziedziczona po WindowAdapter, uruchamiana w przypadku żądania zamknięcia programu */
        public void windowClosing(WindowEvent event)
         {
         Program.komponentSterowanie().zadanieZamknieciaProgramu(gui);
         }


}

```

```java
package pl.nalazek.komunikator.gui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import pl.nalazek.komunikator.Program;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca zarządce obsługi menu */
public class OknoMenu implements ActionListener, ItemListener {
        private Gui gui;

        /** Konstruktor
         * @param gui Referencja do interfejsu graficznego programu */
        public OknoMenu(Gui gui)
        {
        this.gui = gui;
        }

        /** Funkcja realizująca interfejs ActionListener, uruchamiana na skutek zdarzenia przychodzące z menu *
         * @param arg0 Referencja do zdarzenia */
        public void actionPerformed(ActionEvent arg0) {
            if(arg0.getActionCommand() == "Online")
            {
                if(gui.menuOnline.isSelected() == true)
                    Program.komponentSterowanie().zmienionoNaOnline(gui, gui.menuOnlineWybor);
                else
                    Program.komponentSterowanie().zmienionoNaOffline(gui);
            }
            if(arg0.getActionCommand() == "Zako\u0144cz")
            {
                Program.komponentSterowanie().zadanieZamknieciaProgramu(gui);
            }
            if(arg0.getActionCommand() == "O programie...")
            {
                gui.oknoDialogoweInfo();
            }

        }

        /** Funkcja realizująca interfejs ItemListener, uruchamiana na skutek zmiany wyboru trybu w jakim program
  pracuje online *
         * @param arg0 Referencja do zdarzenia */
        public void itemStateChanged(ItemEvent arg0) {
            if(arg0.getSource() == gui.onlnInternet)
            {
                if(gui.onlnInternet.isSelected())
                gui.menuOnlineWybor =  0;
            }
            else if(arg0.getSource() == gui.onlnLchst1)
            {
                if(gui.onlnLchst1.isSelected())
                    gui.menuOnlineWybor =  1;
            }
            else if(arg0.getSource() == gui.onlnLchst2)
            {
                if(gui.onlnLchst2.isSelected())
                    gui.menuOnlineWybor =  2;
            }
        }

}

```

```java
package pl.nalazek.komunikator.gui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JDialog;
import javax.swing.SwingUtilities;
import pl.nalazek.komunikator.Program;
import pl.nalazek.komunikator.logika.Watek;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca zarządce obsługi panelu rozmowa */
public class OknoPanelRozmowa implements ActionListener{
    private Gui gui;

    /** Konstruktor
     * @param gui Referencja do interfejsu graficznego programu */
    public OknoPanelRozmowa(Gui gui)
    {
        this.gui = gui;
        JDialog.setDefaultLookAndFeelDecorated(true);
    }

    /** Funkcja realizująca interfejs ActionListener, uruchamiana na skutek zdarzenia przychodzące z panelu rozmowa *
     * @param arg0 Referencja do zdarzenia */
    public void actionPerformed(ActionEvent arg0) {
        if(arg0.getActionCommand() == "Czy\u015b\u0107")
        {
            SwingUtilities.invokeLater(new Runnable()
            {
                public void run()
                {
                        gui.panel1PoleTekstoweWpissc.setText("");
                }
            });
        }

        else if(arg0.getActionCommand() == "Wy\u015blij")
                Program.komponentSterowanie().zadanieWyslaniaWiadomosci(gui);

        else if(arg0.getActionCommand() == "Wy\u015blij plik")
            (new Watek(){public void run(){
                Program.komponentSterowanie().zadanieWyslaniaPliku(gui);
            }}).start();

    }

}

```

```java
package pl.nalazek.komunikator.gui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JComponent;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import pl.nalazek.komunikator.Program;
import pl.nalazek.komunikator.logika.Watek;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca zarządce obsługi panelu rozmowa */
public class OknoPanelUzytkownicy implements ActionListener, ListSelectionListener{
        private Gui gui;

        /** Konstruktor
         * @param gui Referencja do interfejsu graficznego programu */
        public OknoPanelUzytkownicy(Gui gui)
        {
            this.gui = gui;
            JDialog.setDefaultLookAndFeelDecorated(true);
        }

        /** Funkcja realizująca interfejs ActionListener, uruchamiana na skutek zdarzenia przychodzące z panelu użytkownicy *
         * @param arg0 Referencja do zdarzenia */
        public void actionPerformed(ActionEvent arg0)
        {
            if(arg0.getActionCommand() == "Dodaj")
            {
                SwingUtilities.invokeLater(new Runnable()
                {
                    public void run()
                    {
                    String[] dane = oknoDialogoweNowyUzytkownik();
                    if(dane != null) Program.komponentSterowanie().dodanoUzytkownikaZdalnego(dane);
                    }
                });
            }
            else if(arg0.getActionCommand() == "Usu\u0144")
            {
                    if(!gui.listaUzytkownikow.isSelectionEmpty())
                        (new Watek(){public void run(){
                            Program.komponentSterowanie().usunietoUzytkownikaZdalnego(gui);
                        }}).start();
            }
            else if(arg0.getActionCommand() == "Po\u0142\u0105cz")
            {
                    if(!gui.listaUzytkownikow.isSelectionEmpty())
                    {
                        (new Watek(){public void run(){
                            Program.komponentSterowanie().zadaniePolaczenia(gui);
                        }}).start();

                    }

            }
            else if(arg0.getActionCommand() == "Roz\u0142\u0105cz")
            {

                    if(!gui.listaUzytkownikow.isSelectionEmpty())
                    (new Watek(){public void run(){
                    Program.komponentSterowanie().zadanieRozlaczenia(gui);
                    }}).start();
            }
        }

        /** Funkcja realizująca interfejs ListSelectionListener, uruchamiana na skutek zdarzenia przychodzące z panelu użytkownicy *
```

```
77          * @param arg0 Referencja do zdarzenia */
78         public void valueChanged(ListSelectionEvent arg0) {
79             for(int i = 0; i <= arg0.getLastIndex(); i++)
80             {
81                 if(gui.listaUzytkownikow.isSelectedIndex(i))
82                     gui.listaOstatniIndeks = i;
83             }
84 
85             Program.komponentSterowanie().zmienionoStanListy(gui);
86         }
87 
88         /** Funkcja uruchamia okno dialogowe do wpisywania danych nowego użytkownika */
89         private String[] oknoDialogoweNowyUzytkownik()
90         {
91 
92             JTextField nazwa = new JTextField();
93             JTextField ip = new JTextField();
94             JComponent[] wejscie = new JComponent[]
95                     {
96                     new JLabel("Nazwa u\u017cytkownika: "), nazwa,
97                     new JLabel("Adres IP (dla trybu lokalnego \'localhost\'): "), ip
98                     };
99             if(JOptionPane.showConfirmDialog(gui, wejscie, "Wprowad\u017a dane", JOptionPane.OK_CANCEL_OPTION) ==
   JOptionPane.OK_OPTION)
100             {
101             String ipp = ip.getText();
102             boolean zlaWartosc = false;
103 
104 
105             if(ipp.length() == 15)
106             {
107             ipp = ipp.substring(0, 15);
108                 if(ipp.charAt(3) == '.' &&  ipp.charAt(7) == '.' && ipp.charAt(11) == '.')
109                 {
110                     for(int i = 0; i < ipp.length(); i++ )
111                         {
112                         if(i == 3 || i==7 || i==11) continue;
113                         if(ipp.charAt(i) > 47 && ipp.charAt(i) < 58)
114                             continue;
115                         else zlaWartosc = true;
116                         }
117 
118                 }
119                 else zlaWartosc = true;
120             }
121             else zlaWartosc = true;
122             if(nazwa.getText().length() == 0) {
123                                                 JOptionPane.showMessageDialog(gui, "Wprowad\u017a nazwę
   u\u017cytkownika!",
124                                                 "Wprowadzono b\u0119\u0119dne dane", JOptionPane.WARNING_MESSAGE);
125                                                 return null;
126             }
127 
128             if(ipp.matches("localhost")) return new String[] { nazwa.getText(), "localhost" };
129             if(zlaWartosc)
130                 {
131                 JOptionPane.showMessageDialog(gui, "Wprowadzono adres ip o b\u0142\u0119dnym formacie."
132                         + "\nPoprawny format to xxx.xxx.xxx.xxx gdzie \'x\' oznacza liczb\u0119",
133                         "Wprowadzono b\u0119\u0119dne dane", JOptionPane.WARNING_MESSAGE);
134                 return null;
135                 }
136 
137             return new String[] { nazwa.getText(), ip.getText() };
138             }
139             return null;
140         }
141 
142 
143 }
144 
145 
```

```java
package pl.nalazek.komunikator.logika;

import java.net.InetSocketAddress;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashSet;
import pl.nalazek.komunikator.logika.logowanie.UzytkownikZdalny;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa obsługująca kontakty dla aktualnego użytownika zalogowanego w programie */
public class AktywneKontakty {

    private LinkedHashSet<UzytkownikZdalny> kontakty;
    private HashMap<String,LinkedHashSet<UzytkownikZdalny>> kontaktyIp;
    private HashMap<String,LinkedHashSet<UzytkownikZdalny>> kontaktyNazwa;
    private LinkedHashSet<String[]> kontaktyNazwy;

    /** Konstruktor klasy
     * @param kontakty Kontakty w postaci LinkedHashSet<UzytkownikZdalny> na których klasa będzie pracować
     */
    public AktywneKontakty(LinkedHashSet<UzytkownikZdalny> kontakty)
    {
        this.kontakty = kontakty;
        przebuduj();
    }

    /** Funkcja sprawdza czy dany kontakt istnieje w spisie
     * @param ip IP kontaktu
     * @param nazwa Nazwa użytkownika
     * @return Wartość "true" w przypadku gdy kontakt istnieje, wartość "false" w przypadku gdy kontakt nie istnieje w spisie
     */
    public boolean sprawdzCzyIstnieje(String ip, String nazwa)
    {
        if(kontaktyIp.containsKey(ip))
        {
            Iterator<UzytkownikZdalny> i = kontaktyIp.get(ip).iterator();
            while(i.hasNext())
            {
                if(i.next().zwrocNazwe() == nazwa) return true;
            }
        }
        return false;
    }

    /** Funkcja szuka uzykownika zdalnego w spisie po adresie ip i jego nazwie
     * @param ip IP szukanego użytkownika
     * @param nazwa Nazwa szukanego użytkownika
     * @return Referencja do UzykownikaZdalnego w przypadku odnalezienia, "null" w przypadku nie odnalezienia.
     */
    public UzytkownikZdalny zwrocUzytkownikaZdalnego(String ip, String nazwa)
    {
        if(kontaktyIp.containsKey(ip))
        {
            LinkedHashSet<UzytkownikZdalny> list = kontaktyIp.get(ip);
            Iterator<UzytkownikZdalny> i = list.iterator();
            while(i.hasNext())
            {
                UzytkownikZdalny uZ = i.next();
                if(uZ.zwrocNazwe().equals(nazwa))
                    return uZ;
            }
        }
        return null;
    }

    /** Dodaje nowy kontakt
     * @param adresSocket Adres gniazdka użytkownika zdalnego
     * @param nazwa Nazwa użytkowika zdalnego
     * @return Referencja do obiektu UzytkownikZdalny, gdy kontakt został prawidłowo dodany, wartość "null" gdy kontakt już istnieje
     */
    public UzytkownikZdalny dodajKontakt(InetSocketAddress adresSocket, String nazwa)
    {
```

```
77          if(!sprawdzCzyIstnieje(adresSocket.getAddress().getHostAddress(),nazwa))
78          {
79              UzytkownikZdalny uZ = new UzytkownikZdalny(adresSocket,nazwa);
80              kontakty.add(uZ);
81              przebuduj();
82              return uZ;
83          }
84          return null;
85      }
86
87      /** Zwraca listę kontaktów w postaci tablicy String
88       * @return Duplikat listy kontaktów: String[3] w postaci [0]:nazwa, [1]:ip, [2]:nr użytkownika w systemie
89       */
90      public LinkedHashSet<String[]> zwrocKontaktyNazwy()
91      {
92          return new LinkedHashSet<String[]>(kontaktyNazwy);
93      }
94
95      /** Usuwa kontakt ze spisu *
96       * @param ip Ip użytkownika zdalnego
97       * @param nazwa Nazwa użytkownika zdalnego
98       * @return Wartość "true" gdy kontakt został usunięty, wartość "false" gdy kontakt nie istnieje
99       */
100     public boolean usunKontakt(String ip, String nazwa)
101     {
102         UzytkownikZdalny uZ = zwrocUzytkownikaZdalnego(ip,nazwa);
103         if(uZ != null)
104         {
105             kontakty.remove(uZ);
106             przebuduj();
107             return true;
108         }
109         return false;
110     }
111
112     /** Tworzy listy kontaktów na potrzeby klasy AktywneKontakty */
113     private void przebuduj()
114     {
115         utworzKontaktyIp();
116         utworzKontaktyNazwa();
117         utworzKontaktyNazwy();
118     }
119
120     /** Tworzy odwzorowanie HashMap adresów ip z użytkownikami zdalnymi */
121     private void utworzKontaktyIp()
122     {
123         Iterator<UzytkownikZdalny> i = kontakty.iterator();
124         kontaktyIp = new HashMap<String,LinkedHashSet<UzytkownikZdalny>>();
125
126         while(i.hasNext())
127         {
128             String ip = i.next().zwrocIp();
129             kontaktyIp.put(ip, new LinkedHashSet<UzytkownikZdalny>());
130             Iterator<UzytkownikZdalny> j = kontakty.iterator();
131             while(j.hasNext())
132             {
133                 UzytkownikZdalny uZ = j.next();
134                 if(uZ.zwrocIp() == ip)
135                     kontaktyIp.get(ip).add(uZ);
136             }
137         }
138     }
139
140     /** Tworzy odwzorowanie HashMap nazw użytkowników z użytkownikami zdalnymi */
141     private void utworzKontaktyNazwa()
142     {
143         Iterator<UzytkownikZdalny> i = kontakty.iterator();
144         kontaktyNazwa = new HashMap<String,LinkedHashSet<UzytkownikZdalny>>();
145
146         while(i.hasNext())
147         {
148             String nazwa = i.next().zwrocNazwe();
149             kontaktyNazwa.put(nazwa, new LinkedHashSet<UzytkownikZdalny>());
150             Iterator<UzytkownikZdalny> j = kontakty.iterator();
151             while(j.hasNext())
152             {
153                 UzytkownikZdalny uZ = j.next();
154                 if(uZ.zwrocNazwe() == nazwa)
```

```
155                     kontaktyNazwa.get(nazwa).add(uZ);
156                 }
157             }
158     }
159 
160     /** Tworzy zbiór LinkedHashSet zawierający tablice obiektów String z nazwą użytkownika, adresem ip i nr użytkownika */
161     private void utworzKontaktyNazwy()
162     {
163         Iterator<UzytkownikZdalny> i = kontakty.iterator();
164         kontaktyNazwy = new LinkedHashSet<String[]>();
165 
166         while(i.hasNext())
167         {
168             UzytkownikZdalny uZ = i.next();
169             String[] obj = new String[3];
170             obj[0] = uZ.zwrocNazwe();
171             obj[1] = uZ.zwrocIp();
172             obj[2] = uZ.zwrocNrUzytkownika().toString();
173             kontaktyNazwy.add(obj);
174 
175         }
176     }
177 
178 }
179 
```

```java
package pl.nalazek.komunikator.logika;

import java.io.IOException;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca pojedyńczą paczkę danych w programie */
public class Dane extends DanePytanie {
    private static final long serialVersionUID = 1L;
    private Plik plik;

    /** Konstruktor pakietu danych
     * @param sciezkaPliku Sciezka pliku do wyslania
     * @throws Wyjatek
     * @throws IOException
     * */
    public Dane(RozmowaID rozmowaZrodlowa, RozmowaID rozmowaDocelowa, String sciezkaPliku) throws IOException, Wyjatek
    {
        nazwa = "Pakiet Danych";
        nrid = NrIdPakietu.DANE;
        this.rozmowaDocelowa = rozmowaDocelowa;
        this.rozmowaZrodlowa = rozmowaZrodlowa;
        plik = new Plik(sciezkaPliku);
        plikInfo = (PlikNaglowek)plik;
    }

    /** Zwaraca plik
     * @return Plik zawarty w paczce
     */
    public Plik zwrocPlik()
    {
        return plik;
    }

}
```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca pakiet z odpowiedzią na żądanie danych */
public class DaneOdpowiedz extends PakietOdpowiedz {

    private static final long serialVersionUID = 1L;

    /** Konstruktor */
    DaneOdpowiedz()
    {
        nazwa = "Pakiet Danych Odpowiedz";
        nrid = NrIdPakietu.DANE_ODPOWIEDZ;
    }

    /** Konstruktor *
     * @param pytanie Pytanie na które ten pakiet będzie odpowiadał
     * @param odpowiedz Odpowiedź na pytanie. Wartość "true" oznacza zgodę, wartość "false" brak zgody
     */
    DaneOdpowiedz(DanePytanie pytanie, boolean odpowiedz)
    {
        this();
        rozmowaZrodlowa = pytanie.zwrocIdRozmowyDocelowej();
        rozmowaDocelowa = pytanie.zwrocIdRozmowyZrodlowej();
        if(odpowiedz)
        kluczWeryf = pytanie.kluczWeryf / pytanie.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy();
        else kluczWeryf = -1;
    }

}
```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca pakiet z żądaniem odebrania danych */
public class DanePytanie extends PakietPytanie {

    private static final long serialVersionUID = 1L;
    protected PlikNaglowek plikInfo;

    /** Konstruktor */
    public DanePytanie()
    {
        nazwa = "Pakiet Danych Pytanie";
        nrid = NrIdPakietu.DANE_PYTANIE;
    }

    /** Konstruktor
     * @param dane Paczka danych która ma zostać wysłana
     */
    public DanePytanie(Dane dane)
    {
        this();
        plikInfo = dane.plikInfo;
        rozmowaDocelowa = dane.rozmowaDocelowa;
        rozmowaZrodlowa = dane.rozmowaZrodlowa;
    }

    /** Zwraca informację o pliku
     * @return Informacja o pliku w postaci klasy PlikNaglowek
     */
    public PlikNaglowek zwrocPlikInfo()
    {
        return plikInfo;
    }

}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

import java.util.*;
import java.io.*;
import java.net.InetSocketAddress;
import pl.nalazek.komunikator.logika.logowanie.UzytkownikZdalny;

/** Interfejs obsługujący Model aplikacji*/
public interface IModel {

    /** Zarządza widzialnością użytkownika w sieci
     * @param wartosc Wartość "prawda" umożliwia nawiązanie połącznia z programem z zewnątrz. Wartość "fałsz" wyłącza widoczność w sieci.
     * @param localhost Parametr określający tryb pracy online.
     * */
    void ustawOnline(boolean wartosc, LocalHost localhost) throws IOException, ClassNotFoundException, Wyjatek;

    /** Dodaje uzytkownika zdalnego do listy uzytkownikow obecnie zalogowanego uzytkownika lokalnego.
     * @param nazwaUzytkownika Nazwa (alias) dodawanego uzytkownika.
     * @param adresSocket Gniazdo użytkownika zdalnego
     * */
    UzytkownikZdalny dodajUzytkownikaZdalnego(String nazwaUzytkownika, InetSocketAddress adresSocket) throws Wyjatek;

    /** Usuwa uzytkownika zdalnego z listy uzytkownikow obecnie zalogowanego uzytkownika lokalnego.
     * @param nazwaUzytkownika Nazwa (alias) dodawanego uzytkownika.
     * @param adres Adres IP lub ID z zewnętrznego serwera WWW/SQL.
     * */
    void usunUzytkownikaZdalnego(String nazwaUzytkownika, String adres) throws Wyjatek;

    /** Zwraca adres IP komputera na którym uruchomiony jest program
     * @return Adres IP w postaci ciągu znaków w formacie: xxx.xxx.xxx.xxx
     * */
    String mojeIP();

    /** Wysyła żądanie nawiązania rozmowy ze zdalnym użytkownikiem
     * @param uzytkownik Referencja do uzytkownika zdalnego
     * @return Referencja do rozmowy w przypadku zgody, referencja do rozmowy z id=0 w przypadku odmowy, "null" w przypadku braku odpowiedzi.*/
    RozmowaID rozpocznijRozmowe(UzytkownikZdalny uzytkownik) throws Wyjatek;

    /** Zwraca zbiór użytkowników zdalnych dla obecnie zalogowanego użytkownika lokalnego.
     * @return Lista użytkowników zdalnych w postaci zbioru LinkedHashSet<String[]>: [0]:nazwa użytkownika zdalnego, [1]:ip, [2]:nr użytkownika w systemie */
    LinkedHashSet<String[]> listaUZ();

    /** Wysyła żądanie odebrania pliku przez użytkownika zdalnego
     * @param uZdalny Adresat
     * @param id Referencja do rozmowy
     * @param sciezka_pliku Scieżka wysyłanego pliku
     * @return Obiekt reprezentujący stan wysyłania pliku */
    StanWyslaniaPliku wyslijPlik(UzytkownikZdalny uZdalny, RozmowaID id, String sciezka_pliku) throws Wyjatek;

    /** Wysyła wiadomość do użytkownika
     * @param id Referencja do rozmowy z której wysyłana jest wiadomość
     * @param wiadomosc Treść wiadomości
     * */
    void wyslijWiadomosc(RozmowaID id, String wiadomosc) throws IOException;

    /** Kończy rozmowę
     * @param id Referencja do rozmowy
     */
    void zakonczRozmowe(RozmowaID id) throws IOException;

}
```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Interfejs obsługujący Sterowanie aplikacji */
public interface ISterowanie {

	/** Wysyła żądanie odebrania wiadomosci przez Sterowanie
	 * @param id Identyfikator rozmowy docelowej w programie
	 * @param wiadomosc Treść przesyłanej wiadomości
	 * @param ip Adres Ip nadawcy
	 * */
	void odbierzWiadomosc(String ip, RozmowaID id, String wiadomosc);

	/** Wysyła żądanie odebrania pliku przez Sterowanie
	 * @param ip Adres Ip nadawcy
	 * @param id Identyfikator rozmowy w programie
	 * @param plik Nagłówek przesyłanego pliku
	 * @return Kod odbioru: "0" odmowa, "1" zgoda, "-1" brak odpowiedzi, scieżka pliku
	 * */
	OdpowiedzSterowania odbierzPlik(String ip, RozmowaID id, PlikNaglowek plik, StanWyslaniaPliku postep);

	/** Wysyła żądanie odebrania rozmowy przez Sterowanie
	 * @param ip Adres Ip nadawcy
	 * @param rozmowaId Identyfikator rozmowy w programie
	 * @return Kod odbioru: "0" odmowa, "1" zgoda, "-1"
	 * */
	OdpowiedzSterowania odbierzRozmowe(String ip, RozmowaID rozmowaId);

	/** Wysyła żądanie odebrania zakończenia rozmowy przez Sterowanie
	 * @param ip Adres Ip nadawcy
	 * @param id Identyfikator rozmowy w programie
	 */
	void odbierzZakonczenieRozmowy(String ip, RozmowaID id);

	/** Wysyła żądanie odebrania wyjątku przez Sterowanie
	 * @param opis Treść wyjątku
	 */
	void odbierzWyjatek(String opis);

}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Zmienna enum reprezentująca wybór trybu pracy online */
public enum LocalHost {
    IP("Adres w sieci zewnętrznej"), LOCALHOST1("Pierwszy serwer lokalny"), LOCALHOST2("Drugi serwer lokalny");

    private String opis;
    private LocalHost(String opis)
    {
        this.opis = opis;
    }

    /** Tworzy zmienną Localhost z liczby całkowitej
     * @param nr 0:IP, 1:LOCALHOST1, 2:LOCALHOST2
     * @return enum LocalHost
     */
    public static LocalHost fromInt(int nr)
    {
        switch(nr)
        {
        case 0: return LocalHost.IP;
        case 1: return LocalHost.LOCALHOST1;
        case 2: return LocalHost.LOCALHOST2;
        }
        return null;
    }

    /** Zwraca opis zmiennej
     * @return Opis zmiennej
     */
    public String getDescription()
    {
        return opis;
    }

}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

import pl.nalazek.komunikator.Program;
import pl.nalazek.komunikator.logika.logowanie.UzytkownikLokalny;
import pl.nalazek.komunikator.logika.logowanie.UzytkownikZdalny;
import java.io.*;
import java.util.*;
import java.net.*;


/** Klasa odpowiadająca za model programu */
public class Model implements IModel {

    private WatekSluchajacy watekSluchajacy = null;
    static final int portSluch = 5622, portSluchLH1 = 5623, portSluchLH2 = 5624;
    private static int portSluchAktywny;
    private UzytkownikLokalny aktywnyUzytkownikLokalny;
    private AktywneKontakty kontakty;
    /** Nr rozmowy / rozmowa */
    private volatile HashMap<Long,Rozmowa> rozmowy;
    /** Nr rozmowy / odpowiedz */
    private volatile HashMap<Long,Pakiet> pytania;




    /** Konstruktor klasy */
    public Model()
    {
        rozmowy = new HashMap<Long,Rozmowa>();
        pytania = new HashMap<Long,Pakiet>();
        Watek.ustawRefModelu(this);
    }

    public void ustawOnline(boolean wartosc, LocalHost localhost)
            throws IOException, ClassNotFoundException, Wyjatek {
        if(wartosc){
        if(localhost == LocalHost.IP)
        {
            watekSluchajacy = new WatekSluchajacy(portSluchAktywny = portSluch);
            watekSluchajacy.start();
        }
        else if(localhost == LocalHost.LOCALHOST1)
        {
            watekSluchajacy = new WatekSluchajacy(portSluchAktywny = portSluchLH1);
            watekSluchajacy.start();
        }
        else if(localhost == LocalHost.LOCALHOST2)
        {
            watekSluchajacy = new WatekSluchajacy(portSluchAktywny = portSluchLH2);
            watekSluchajacy.start();
        }
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
        }
        if(!watekSluchajacy.isAlive()) throw new Wyjatek("Nie można otworzyć portu do słuchania");
        }

        else
            {
            watekSluchajacy.interrupt();
            portSluchAktywny = 0;
            }


    }

    public String mojeIP() {
        try{ return InetAddress.getLocalHost().toString(); }
        catch(UnknownHostException a) { return "Nieznany host"; }
    }
```

```
 79
 80    public RozmowaID rozpocznijRozmowe(UzytkownikZdalny uzytkownik) throws Wyjatek {
 81        Rozmowa rozmowa = new Rozmowa(aktywnyUzytkownikLokalny,uzytkownik);
 82        if(rozmowa.czyAktywna()) {
 83        RozmowaPytanie pytanie = new RozmowaPytanie(rozmowa.zwrocIdRozmowy());
 84        pytania().put(pytanie.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy(),null);
 85        synchronized(rozmowa.semaforKolejek)
 86        {rozmowa.obslugaKolejki(uzytkownik, pytanie);}
 87
 88
 89        try {
 90            for(int i = 0; i<50; i++)
 91            {
 92                Thread.sleep(200);
 93                /* sprawdzenie czy odpowiedz która przyszła jest opdowiedzią oczekiwaną */
 94                RozmowaOdpowiedz pakietOdpowiadajacy = (RozmowaOdpowiedz)
    pytania().get(pytanie.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy());
 95                if(pakietOdpowiadajacy != null)
 96                    {
 97                    /* Odpowiedź pozytywna */
 98                    if(pakietOdpowiadajacy.sprawdzPoprawnoscOdpowiedzi(pytanie))
 99                            {
100                            rozmowa.dodajIdRozmowyDocelowej(uzytkownik,
    pakietOdpowiadajacy.zwrocIdRozmowyZrodlowej());
101                            rozmowy().put(rozmowa.zwrocIdRozmowy().zwrocIdRozmowy(), rozmowa);
102                            pytania().remove(pytanie.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy());
103                            return rozmowa.zwrocIdRozmowy();
104                            }
105                    /* Odpowiedź negatywna */
106                    else
107                            {
108                            pytania().remove(pytanie.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy());
109                            rozmowa.usunUzytkownikaZdalnego(uzytkownik, false);
110                            return new RozmowaID(0);
111                            }
112                    }
113            }
114            pytania().remove(pytanie.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy());
115            rozmowa.usunUzytkownikaZdalnego(uzytkownik, false);
116        } catch (InterruptedException e) {
117            throw new Wyjatek("W oczekiwaniu na odpowiedź wystąpił wyjątek.\n" + e.getMessage() + ";;" +
    e.getCause());
118        }   }
119
120
121        return null;
122    }
123
124    public LinkedHashSet<String[]> listaUZ() {
125        return kontakty.zwrocKontaktyNazwy();
126    }
127
128    public StanWyslaniaPliku wyslijPlik(UzytkownikZdalny uZdalny, RozmowaID id, String sciezka_pliku) throws Wyjatek
    {
129
130        try {
131            Dane dane = new Dane(id, null, sciezka_pliku);
132            DanePytanie pytanie = new DanePytanie(dane);
133            pytania().put(pytanie.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy(),null);
134            synchronized(rozmowy().get(id.zwrocIdRozmowy()).semaforKolejek)
135            {rozmowy.get(id.zwrocIdRozmowy()).obslugaKolejki(uZdalny, pytanie);}
136
137            for(int i = 0; i<8; i++)
138            {
139                Thread.sleep(2000);
140                /* sprawdzenie czy odpowiedz która przyszła jest opdowiedzią oczekiwaną */
141                DaneOdpowiedz pakietOdpowiadajacy = (DaneOdpowiedz)
    pytania().get(pytanie.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy());
142                if(pakietOdpowiadajacy != null)
143                    {
144                    /* Odpowiedź pozytywna */
145                    if(pakietOdpowiadajacy.sprawdzPoprawnoscOdpowiedzi(pytanie))
146                            {
147                        synchronized(rozmowy().get(id.zwrocIdRozmowy()).semaforKolejek)
148                            {rozmowy.get(id.zwrocIdRozmowy()).obslugaKolejki(uZdalny, dane);}
149                            pytania().remove(dane.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy());
150                            return new StanWyslaniaPliku(1);
151                            }
```

```
152                     /* Odpowiedź negatywna */
153                     else
154                             {
155                             pytania().remove(dane.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy());
156                             return new StanWyslaniaPliku(0);
157                             }
158                     }
159             }
160             pytania().remove(dane.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy());
161             return new StanWyslaniaPliku(-1);
162         }
163         catch (InterruptedException e)
164         {
165         throw new Wyjatek("W oczekiwaniu na odpowiedź wystąpił wyjątek.\n" + e.getMessage() + ";;" + e.getCause());
166         }
167         catch (IOException e)
168         {
169         Program.komponentSterowanie().odbierzWyjatek(e.getMessage() + " " + e.getCause());
170         }
171         catch (Wyjatek e)
172         {
173         Program.komponentSterowanie().odbierzWyjatek(e.getMessage() + " " + e.getCause());
174         }
175         return null;
176     }
177
178     public void wyslijWiadomosc(RozmowaID id, String wiadomosc)
179             throws IOException {
180         Wiadomosc wiad = new Wiadomosc(wiadomosc, id);
181         synchronized(rozmowy.get(id.zwrocIdRozmowy()).semaforKolejek)
182         {rozmowy.get(id.zwrocIdRozmowy()).obslugaKolejki(null, wiad);}
183
184     }
185
186     public void zakonczRozmowe(RozmowaID id) throws IOException {
187
188         rozmowy.get(id.zwrocIdRozmowy()).zakoncz();
189     }
190
191     /** Zwraca aktywny port słuchający
192      * @return 5622 w trybie Internet, 5623 w trybie LocalHost1, 5624 w trybie LocalHost2;
193      */
194     int zwrocAktywnyPortSluchajacy()
195     {
196         return portSluchAktywny;
197     }
198
199
200     /** Loguje użytkownika lokalnego do modelu programu *
201      * @param uzytkowikLokalny Użytkownik lokalny która ma zostać zalogowany
202      * @param kontakty Lista kontaktow w postaci zbioru LinkedHashSet<UzytkownikZdalny>
203      */
204     public void zalogowano(UzytkownikLokalny uzytkowikLokalny, LinkedHashSet<UzytkownikZdalny> kontakty)
205     {
206         aktywnyUzytkownikLokalny = uzytkowikLokalny;
207         this.kontakty = new AktywneKontakty(kontakty);
208     }
209
210     /** Wylogowywuje użytkownika lokalnego */
211     public void wylogowano()
212     {
213         aktywnyUzytkownikLokalny = null;
214         this.kontakty = null;
215     }
216
217     /** Zwraca zalogowanego użytkownmika lokalnego
218      * @return Zalogowany użytkownik lokalny
219      */
220     public UzytkownikLokalny zwrocAktywnyUzytkownikLokalny()
221     {
222         return aktywnyUzytkownikLokalny;
223     }
224
225     //metody synchronizowane
226     /** Dodaje użytkownika zdalnego
227      * @param nazwa Nazwa użytkownika zdalnego
228      * @param adresSocket Adres gniazdka
229      * @return Dodany użytkownik zdalny
```

```
230     */
231    synchronized public UzytkownikZdalny dodajUzytkownikaZdalnego(String nazwa, InetSocketAddress adresSocket) throws
   Wyjatek
232    {
233        UzytkownikZdalny uZ = kontakty.dodajKontakt(adresSocket, nazwa);
234        if(uZ == null) throw new Wyjatek("Kontakt już istnieje!");
235        else return uZ;
236    }
237
238    /** Usuwa użytkownika zdalnego
239     * @param nazwa Nazwa użytkownika zdalnego
240     * @param ip Adres ip użytkownika
241     */
242    synchronized public void usunUzytkownikaZdalnego(String nazwa, String ip) throws Wyjatek
243    {
244        if(!kontakty.usunKontakt(ip, nazwa)) throw new Wyjatek("Kontakt nie istnieje!");
245    }
246
247    /** Zwraca użytkownika zdalnego *
248     * @param nazwa Nazwa użytkownika zdalnego
249     * @param ip Adres ip użytkownika
250     * @return W przypadku odnalezienia zwraca użytkownika zdalnego, w przeciwnym razie "null"
251     */
252    synchronized public UzytkownikZdalny zwrocUzytkownikaZdalnego(String ip, String nazwa)
253    {
254        return kontakty.zwrocUzytkownikaZdalnego(ip, nazwa);
255    }
256
257    /** Zwraca przeciwny port słuchający
258     * @return 5622 w trybie Internet, 5624 w trybie LocalHost1, 5623 w trybie LocalHost2
259     */
260    synchronized static int zwrocAktywnyPortSluchajacyPrzeciwny()
261    {
262        if(portSluchAktywny == portSluch) return portSluch;
263        else if (portSluchAktywny == portSluchLH1) return portSluchLH2;
264        else if (portSluchAktywny == portSluchLH2) return portSluchLH1;
265        else return -1;
266    }
267
268    /** Umożliwia synchronizowany dostęp do toczących się w programie rozmów
269     * @return Odwzorowanie w postaci nr id rozmowy i referencji do rozmowy
270     */
271    synchronized HashMap<Long,Rozmowa> rozmowy()
272    {
273        return rozmowy;
274    }
275
276    /** Umożliwia synchronizowany dostęp do aktywnych pytań w programie
277     * @return Odwzorowanie w postaci nr id rozmowy i referencji pakietu odpowiadającego
278     */
279    synchronized HashMap<Long,Pakiet> pytania()
280    {
281        return pytania;
282    }
283
284
285
286 }
```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Zmienna enum reprezentująca rodzaj przesyłanego pakietu */
public enum NrIdPakietu {

    PAKIET(100), WIADOMOSC(101), DANE(102), DANE_PYTANIE(105), DANE_ODPOWIEDZ(106), ROZMOWA_PYTANIE(110),
    ROZMOWA_ODPOWIEDZ(111), ROZMOWA_KONIEC(112);

    private int wartosc;

    private NrIdPakietu(int wartosc) {
        this.wartosc = wartosc;
     }

    /** Zwraca rodzaj pakietu
     * @param x Liczba całkowita określająca nr pakietu
     * @return Nazwę pakietu
     */
    public static NrIdPakietu fromInteger(int x)
    {
        switch(x){
        case 100: return PAKIET;
        case 101: return WIADOMOSC;
        case 102: return DANE;
        case 105: return DANE_PYTANIE;
        case 106: return DANE_ODPOWIEDZ;
        case 110: return ROZMOWA_PYTANIE;
        case 111: return ROZMOWA_ODPOWIEDZ;
        case 112: return ROZMOWA_KONIEC;
        default: return null;
        }

    }

    /** Zwraca wartość klasy w postaci liczbowej */
    public int toInt()
    {
        return wartosc;
    }

}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca komunikat wysyłany przez Sterowanie do Modelu */
public class OdpowiedzSterowania {
    public Integer kod;
    public String sciezka;

    /** Konstruktor
     * @param kod Kod jaki przyjmuje klasa
     */
    public OdpowiedzSterowania(Integer kod)
    {
        this.kod = new Integer(kod);
    }

    /** Konstruktor
     * @param kod Kod jaki przyjmuje klasa
     * @param sciezka Ścieżka pliku wybrana przez użytkownika lokalnego.
     * Konstruktor używany w przypadku użycia klasy OdpowiedzSterowania do wygenerowania odpowiedzi na żądanie danych
     */
    public OdpowiedzSterowania(Integer kod, String sciezka)
    {
        this.kod =  new Integer(kod);
        this.sciezka = new String(sciezka);
    }
}

```

```java
package pl.nalazek.komunikator.logika;

import java.io.*;
import java.util.Random;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca pakiet danych przesyłanych między instancjami programu *
 */
public class Pakiet implements Serializable {

    static final long serialVersionUID = 1L;
    String nazwa = "Pakiet Ogolny";
    protected RozmowaID rozmowaZrodlowa, rozmowaDocelowa;
    protected long kluczWeryf;
    NrIdPakietu nrid = NrIdPakietu.PAKIET;

    /** Konstruktor */
    public Pakiet()
    {
        Random generator = new Random();
        kluczWeryf = generator.nextLong();
    }

    /** Zwraca nazwę pakietu
     * @return Nazwa pakietu w postaci ciągu znaków
     */
    public String getNazwa()
    {
        return nazwa;
    }

    /** Zwraca NrIdPakietu
     * @return NrIdPakietu
     */
    public NrIdPakietu getNrId()
    {
        return nrid;
    }

    /** Zwraca referencję do rozmowy źródłowej, tzn. nadawcy pakietu *
     * @return Referencja do składnika rozmowa źródłowa.
     */
    public RozmowaID zwrocIdRozmowyZrodlowej()
    {
        return rozmowaZrodlowa;
    }

    /** Zwraca referencję do rozmowy docelowej, tzn. adresata pakietu *
     * @return Referencja do rozmowy docelowej
     */
    public RozmowaID zwrocIdRozmowyDocelowej()
    {
        return rozmowaDocelowa;
    }

    /** Zwraca klucz weryfikacyjny pakietu
     * @return Klucz weryfikacyjny w postaci long
     */
    public long zwrocKlucz()
    {
        return kluczWeryf;
    }

    /** Dodaje Id rozmowy docelowej *
     * @param rozmowaDocelowa Id rozmowy docelowej
     */
    void dodajIdRozmowyDocelowej(RozmowaID rozmowaDocelowa)
    {
        this.rozmowaDocelowa = rozmowaDocelowa;
    }
}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Abstrakcyjna klasa reprezentująca odpowiedź na pakiety pytajace */
public abstract class PakietOdpowiedz extends Pakiet {
    private static final long serialVersionUID = 1L;

    /** Weryfikuje poprawność odpowiedzi
     * @param pytanie Pakiet pytający
     * @return Wartość "true" w przypadku potwierdzenia poprawności odpowiedzi, wartość "false" w przeciwnym wypadku
     */
    public boolean sprawdzPoprawnoscOdpowiedzi(PakietPytanie pytanie)
    {
        long t = pytanie.kluczWeryf / pytanie.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy();
        if(t == kluczWeryf) return true;
        else return false;
    }


}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Abstrakcyjna klasa reprezentująca pakiet pytający */
public abstract class PakietPytanie extends Pakiet  {

private static final long serialVersionUID = 1L;

}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

public class PakietTX extends Pakiet {
    private static final long serialVersionUID = 1L;
    public String komunikat;
}

```

```java
package pl.nalazek.komunikator.logika;

import java.io.*;
import java.lang.management.ManagementFactory;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezenująca plik danych */
public class Plik extends PlikNaglowek  {

    private static final long serialVersionUID = 1L;
    byte[] dane;

    /** Konstruktor
     * @param sciezka_pliku Ścieżka pliku
     * @throws IOException
     * @throws Wyjatek
     */
    public Plik(String sciezka_pliku) throws IOException, Wyjatek
    {
        danePliku = new File(sciezka_pliku);
        FileInputStream fis = new FileInputStream(danePliku);
        long free = ManagementFactory.getMemoryMXBean().getHeapMemoryUsage().getMax() -
ManagementFactory.getMemoryMXBean().getHeapMemoryUsage().getUsed();
        Float free2 = (float) (free/(1000000));
        if(fis.available() < free )
        {
            dane = new byte[ (int) danePliku.length() ];
            fis.read(dane);
        }
        else {
            fis.close();
            throw new Wyjatek("Plik który próbujesz wysłać jest za duży!\n"
                    + "Maksymalny rozmiar pamięci JavaVM to: " + free + "bajtów (" + free2 + "MB)." );

        }
        fis.close();
    }

    /** Zapisuje plik na dysku
     * @param sciezka Ścieżka do której będzie zapisany plik
     * @throws IOException
     */
    public void zapiszPlik(String sciezka) throws IOException
    {
        FileOutputStream fos = new FileOutputStream(sciezka);
        fos.write(dane);
        fos.close();

    }
}

```

```java
package pl.nalazek.komunikator.logika;

import java.io.File;
import java.io.Serializable;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca dane o pliku przechowywanym w klasie Plik */
public class PlikNaglowek implements Serializable {
    private static final long serialVersionUID = 1L;
    public File danePliku;
}

```

```java
package pl.nalazek.komunikator.logika;

import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Queue;
import pl.nalazek.komunikator.Program;
import pl.nalazek.komunikator.logika.logowanie.UzytkownikLokalny;
import pl.nalazek.komunikator.logika.logowanie.UzytkownikZdalny;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca rozmowę w programie */
public class Rozmowa {
    private RozmowaID idRozmowy;
    private UzytkownikLokalny uLokalny;
    private volatile HashMap<UzytkownikZdalny,Queue<Pakiet>> kolejkaWych;
    private volatile HashMap<UzytkownikZdalny,WatekWysylajacy> watkiWysylajacy;
    private volatile HashMap<UzytkownikZdalny,RozmowaID> idRozmowyDocelowej;
    Object semaforKolejek;
    private boolean aktywna = false;

    /** Konstruktor
     * @param uLokalny Użytkownik lokalny inicjujący rozmowę
     * @param uZdalny Użytkownik zdalny z którym rozpoczynana jest rozmowa
     * @throws Wyjatek
     */
    public Rozmowa(UzytkownikLokalny uLokalny, UzytkownikZdalny uZdalny) throws Wyjatek
    {
        Integer[] zmienne = {uLokalny.zwrocNrUzytkownika(), uZdalny.zwrocNrUzytkownika()};
        idRozmowy = new RozmowaID(zmienne);
        idRozmowy.ustawNazweUzytkownika(uLokalny.zwrocNazwe());
        this.uLokalny = uLokalny;
        kolejkaWych = new HashMap<UzytkownikZdalny,Queue<Pakiet>>();
        watkiWysylajacy = new HashMap<UzytkownikZdalny,WatekWysylajacy>();
        idRozmowyDocelowej = new HashMap<UzytkownikZdalny,RozmowaID>();
        semaforKolejek = new Object();
        dodajUzytkownikaZdalnego(uZdalny);

    }

    /** Kończy wszystkie nawiązane połączenia w rozmowie. Wysyła pakiet RozmowaKoniec do uczestników rozmowy */
    public void zakoncz()
    {

        Iterator<UzytkownikZdalny> u = watkiWysylajacy.keySet().iterator();
        while(u.hasNext()) usunUzytkownikaZdalnego(u.next(),true);
        aktywna = false;
    }

    /** Kończy wszystkie nawiązane połączenia w rozmowie. Nie wysyła pakietu RozmowaKoniec do uczestników rozmowy */
    public void zakonczWymuszenie()
    {

        Iterator<UzytkownikZdalny> u = watkiWysylajacy.keySet().iterator();
        while(u.hasNext()) usunUzytkownikaZdalnego(u.next(),false);
        aktywna = false;
    }

    /** Zwraca id rozmowy
     * @return id rozmowy
     */
    synchronized public RozmowaID zwrocIdRozmowy()
    {
        return idRozmowy;
    }

    /** Zwraca użytkownika lokalnego - gospodarza rozmowy w programie
     * @return Użytkownik lokalny
     */
    synchronized public UzytkownikLokalny zwrocUzytkownikaLokalnego()
    {
        return uLokalny;
    }

```

```java
    /** Dodaje użytkownika zdalnego do rozmowy, tworząc nowy wątek wysyłający oraz kolejkę wychodzącą
     * @param uZdalny Dodawany użytkownik zdalny
     * @throws Wyjatek
     */
    synchronized public void dodajUzytkownikaZdalnego(UzytkownikZdalny uZdalny) throws Wyjatek
    {
        int port;
        port = Model.zwrocAktywnyPortSluchajacyPrzeciwny();
        WatekWysylajacy nowyWatek = new WatekWysylajacy(this,uZdalny,port);
        watkiWysylajacy.put(uZdalny, nowyWatek);
        watkiWysylajacy.get(uZdalny).start();
        while(nowyWatek.getState() != Thread.State.WAITING)
            if(nowyWatek.getState() == Thread.State.TERMINATED) {
                break;
            };
        if(nowyWatek.getState() == Thread.State.WAITING) {
            kolejkaWych.put(uZdalny, new LinkedList<Pakiet>());
            aktywna = true;
        }
        else {
            watkiWysylajacy.remove(uZdalny);
            zakoncz();
            throw new Wyjatek("Nie można połączyć ze zdalnym użytkownikiem!");
        }


    }

    /** Usuwa użytkownika zdalnego z rozmowy.
     * @param uZdalny Usuwany użytkownik zdalny
     * @param jaRozlaczam Wartość "true" oznacza, że inicjującym rozłączenie jest użytkownik lokalny. Wartość "false" oznacza, że inicjującym rozłączenie jest użytkownik zdalny
     */
    synchronized public void usunUzytkownikaZdalnego(UzytkownikZdalny uZdalny, boolean jaRozlaczam)
    {
        if(jaRozlaczam)obslugaKolejki(uZdalny, new RozmowaKoniec(idRozmowyDocelowej.get(uZdalny),idRozmowy));
        if(watkiWysylajacy.get(uZdalny) != null)
        while(!( watkiWysylajacy.get(uZdalny).getState() == Thread.State.WAITING ||
watkiWysylajacy.get(uZdalny).getState() == Thread.State.TERMINATED )) ;

        kolejkaWych.remove(uZdalny);
        synchronized(watkiWysylajacy.get(uZdalny).semafor)
        {
            watkiWysylajacy.get(uZdalny).interrupt();
        }
        watkiWysylajacy.remove(uZdalny);
        usunIdRozmowyDocelowej(uZdalny);
    }

    /** Dodaje id rozmowy docelowej dla danego użytkownika zdalnego w celu poprawnego adresowania pakietów
     * @param uZdalny Użytkownik zdalny
     * @param id Dodawane id rozmowy docelowej
     */
    synchronized public void dodajIdRozmowyDocelowej(UzytkownikZdalny uZdalny, RozmowaID id)
    {
        idRozmowyDocelowej.put(uZdalny, id);
    }

    /** Usuwa id rozmowy docelowej dla danego użytkownika zdalnego
     * @param uZdalny Użytkownik zdalny
     */
    synchronized private void usunIdRozmowyDocelowej(UzytkownikZdalny uZdalny)
    {
        idRozmowyDocelowej.remove(uZdalny);
    }

    /** Zwraca informację o tym czy rozmowa jest aktywna, tzn. czy jest połączenie co najmniej z jednym użytkownikiem zdalnym
     * @return Wartość "true" w przypadku gdy rozmowa jest aktywna, wartość "false" w przeciwnym wypadku
     */
    boolean czyAktywna()
    {
        return aktywna;
    }

    /** Obsługuje kolejkę wychodzącą dla danego użytkownika. Wyjmuje lub wkłada pakiety do kolejki, a także informuje wątek wysyłający o nowym pakiecie w kolejce.
     * @param uZdalny Użytkownik zdalny do którego kolejki chcemy się odnieść. Wartość "null" powoduje włożenie
```

```
   pakietu do wszytskich kolejek w rozmowie.
153     * @param pakiet Pakiet do przesłania. Wartość "null" powoduję wyjęcie z kolejki pierwszego elementu (funkcja
   używana przez wątek wysyłający).
154     * @return W przypadku podania jako drugi parametr wartości "null" zwraca Pakiet, w przeciwnym razie zwraca
   wartość "null".
155     */
156    Pakiet obslugaKolejki(UzytkownikZdalny uZdalny, Pakiet pakiet)
157    {
158        if(uZdalny != null)
159        {
160            if(pakiet == null)
161                synchronized(semaforKolejek)
162                { return kolejkaWych.get(uZdalny).poll(); }
163            else
164                {
165                if(pakiet.zwrocIdRozmowyDocelowej() == null)
   pakiet.dodajIdRozmowyDocelowej(idRozmowyDocelowej.get(uZdalny)); //oznaczenie pakietu
166                    try{
167                    synchronized(semaforKolejek)
168                    {kolejkaWych.get(uZdalny).offer(pakiet);}
169                    }
170                    catch(IllegalStateException w)
171                    {
172                    Program.komponentSterowanie().odbierzWyjatek("W rozmowie: " + idRozmowy.zwrocIdRozmowy() + "
   wystąpił wyjątek.\n" + w.getMessage() + ";;" + w.getCause());
173                    }
174                    synchronized (watkiWysylajacy.get(uZdalny).semafor)
175                    {
176                        watkiWysylajacy.get(uZdalny).semafor.notify();
177                        }
178                    return null;
179                }
180        }
181        else {
182            Iterator<UzytkownikZdalny> i;
183            synchronized(semaforKolejek) {i = kolejkaWych.keySet().iterator();}
184                while(i.hasNext())
185                {
186                    UzytkownikZdalny uZ = i.next();
187                    if(pakiet.zwrocIdRozmowyDocelowej() == null)
   pakiet.dodajIdRozmowyDocelowej(idRozmowyDocelowej.get(uZ)); //oznaczenie pakietu
188                    try{
189                        synchronized(semaforKolejek){
190                    kolejkaWych.get(uZ).offer(pakiet);}
191                    }
192                    catch(IllegalStateException w)
193                    {
194                    Program.komponentSterowanie().odbierzWyjatek("W rozmowie: " + idRozmowy.zwrocIdRozmowy() + "
   wystąpił wyjątek.\n" + w.getMessage() + ";;" + w.getCause());
195                    }
196                    synchronized (watkiWysylajacy.get(uZ).semafor)
197                    {
198                        watkiWysylajacy.get(uZ).semafor.notify();
199                        }
200                    return null;
201                }
202            }
203        return null;
204    }
205}
206
```

```java
package pl.nalazek.komunikator.logika;

import java.io.Serializable;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa zawierająca nr identyfikacyjny rozmowy na komputerze lokalnym */
public class RozmowaID implements Serializable {
    private static final long serialVersionUID = 1L;
    private long idRozmowy;
    private String nazwaUzytkownikaLokalnego;

    /** Konstruktor */
    public RozmowaID()
    {
        idRozmowy = System.currentTimeMillis();
    }

    /** Konstruktor z parametrem zmieniającym nr id
     * @param zmienna Tablica zmiennych typu Integer na podstawie których zmieniany jest nr id
     */
    RozmowaID(Integer[] zmienna)
    {
        this();
        for(Integer a : zmienna)
            idRozmowy -= a;
    }

    /** Konstruktor z parametrem
     * @param nr Liczba całkowita którą będzie nr id
     */
    RozmowaID(int nr)
    {
        idRozmowy = nr;
    }

    /** Konstruktor z parametrami
     * @param nr Liczba całkowita którą będzie nr id
     * @param a Nazwa użytkownika lokalnego, który będzie właścicielem rozmowy
     */
    RozmowaID(int nr, String a)
    {
        idRozmowy = nr;
        nazwaUzytkownikaLokalnego = a;
    }

    /** Zwraca id rozmowy w postaci liczbowej
     * @return Liczba całkowita będąca nr id rozmowy
     */
    long zwrocIdRozmowy()
    {
        return idRozmowy;
    }

    /** Zwraca id rozmowy w postaci liczbowej
     * @return Liczba całkowita będąca nr id rozmowy
     */
    long toLong()
    {
        return idRozmowy;
    }

    /** Ustawia nazwę użytkownika, który jest właścicielem rozmowy
     * @param nazwa Nazwa użytkownika
     */
    void ustawNazweUzytkownika(String nazwa)
    {
        nazwaUzytkownikaLokalnego = nazwa;
    }

    /** Zwraca nazwę użytkownika, który jest właścicielem rozmowy
     * @return Nazwa użytkownika, właściciela rozmowy
     */
    String zwrocNazweUzykownika()
    {
```

```
79          return nazwaUzytkownikaLokalnego;
80      }
81  }
82
```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Pakiet wysyłany w przypadku zakończenia rozmowy przez którąś ze stron */
public class RozmowaKoniec extends Pakiet{
        private static final long serialVersionUID = 1L;

        /** Konstruktor pakietu
         * @param rozmowaDocelowa Referencja do rozmowy docelowej
         * @param rozmowaZrodlowa Referencja do rozmowy źródłowej
         */
        public RozmowaKoniec(RozmowaID rozmowaDocelowa, RozmowaID rozmowaZrodlowa)
        {
            nazwa = "Pakiet Rozmowa Koniec";
            nrid = NrIdPakietu.ROZMOWA_KONIEC;
            this.rozmowaDocelowa = rozmowaDocelowa;
            this.rozmowaZrodlowa = rozmowaZrodlowa;
        }
}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca pakiet z opdowiedzią na żądanie rozpoczęcia rozmowy */
public class RozmowaOdpowiedz extends PakietOdpowiedz {
    private static final long serialVersionUID = 1L;



    /** Konstruktor pakietu
     * @param pytanie Referencja do pakietu pytającego
     * @param rozmowaZrodlowa Referencja do utworzonej rozmowy w odpowiedzi na pakiet pytający. Wartość "null" oznacza brak zgody.
     */
    public RozmowaOdpowiedz(RozmowaPytanie pytanie, RozmowaID rozmowaZrodlowa)
    {
        nazwa = "Pakiet Rozmowa Odpowiedz";
        nrid = NrIdPakietu.ROZMOWA_ODPOWIEDZ;
        rozmowaDocelowa = pytanie.zwrocIdRozmowyZrodlowej();
        this.rozmowaZrodlowa = rozmowaZrodlowa;
        if(rozmowaZrodlowa.zwrocIdRozmowy() != 0) kluczWeryf = pytanie.kluczWeryf / pytanie.zwrocIdRozmowyZrodlowej().zwrocIdRozmowy();
        else kluczWeryf = -1;
    }


}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca pakiet z pytaniem o rozpoczęcie rozmowy */
public class RozmowaPytanie extends PakietPytanie {
    private static final long serialVersionUID = 1L;

    /** Konstruktor pakietu
     * @param rozmowaZrodlowa Numer id rozmowy źródłowej, której ma dotyczyć odpowiedź
     */
    public RozmowaPytanie(RozmowaID rozmowaZrodlowa)
    {
        nazwa = "Pakiet Rozmowa Pytanie";
        nrid = NrIdPakietu.ROZMOWA_PYTANIE;
        this.rozmowaZrodlowa = rozmowaZrodlowa;

    }

}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentującą opdowiedź na żądanie wysłania pliku */
public class StanWyslaniaPliku {
    int stan;

    /** Kontruktor pakietu
     * @param stan Liczba całkowita określająca stan odpowiedzi na żądanie wysłania pliku
     */
    public StanWyslaniaPliku(int stan)
    {
        this.stan = stan;
    }
}

```

```java
package pl.nalazek.komunikator.logika;

import java.awt.FileDialog;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.net.InetSocketAddress;
import java.util.Calendar;
import java.util.Iterator;
import java.util.LinkedHashSet;
import javax.swing.SwingUtilities;
import pl.nalazek.komunikator.Program;
import pl.nalazek.komunikator.gui.Gui;
import pl.nalazek.komunikator.logika.logowanie.Logowanie;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa odpowiadająca za sterownie w programie */
public class Sterowanie implements ISterowanie {

    private Gui gui;
    private Sterowanie sterowanie;
    private Model model;
    private Logowanie logowanie;
    private boolean online = false;
    private LinkedHashSet<String[]> listaKluczy;
    private RozmowaID biezacaRozmowa;
    private String zmiennaText;

    /** Konstruktor. Tworzy instancję komponentu Logowanie */
    public Sterowanie()
    {
        logowanie = new Logowanie(this);
    }

    /** Uruchamia sterowanie w programie. */
    public void sterowanieStart()
    {
        sterowanie = this;
        gui = Program.komponentGui(this);
        model = Program.komponentModel(this);

        gui.dezaktywujPanelRozmowa();
        gui.dezaktywujPrzyciskiPaneluUzytkownicy();
        gui.dezaktywujPrzyciskiPaneluUzytkownicyLaczenie();
        logowanie.zaloguj("domyslny", "domyslny");
        odswiezListeKontaktow();
        SwingUtilities.invokeLater(new Runnable(){public void run() { gui.panel2Dodaj.setEnabled(true); }} );
    }

    /** Odświeża listę kontaktów widzialną w głównym oknie programu */
    private void odswiezListeKontaktow()
    {
        String [] t = wygenerujListeUzytkownikow();
        if(t[0] != "")
        SwingUtilities.invokeLater(new Runnable(){public void run() {
  gui.listaUzytkownikow.setListData(wygenerujListeUzytkownikow());
                                                                    gui.listaUzytkownikow.setEnabled(true);
                                                                    gui.panel2Usun.setEnabled(true);}} );
        else SwingUtilities.invokeLater(new Runnable(){public void run() {
  gui.listaUzytkownikow.setListData(wygenerujListeUzytkownikow());
                                                                    gui.listaUzytkownikow.clearSelection();
                                                                    gui.listaUzytkownikow.setEnabled(false);
                                                                    gui.panel2Usun.setEnabled(false);}} );

    }

    /** Tworzy listę kontaktów, która będzie widoczna w głównym oknie programu */
    private String[] wygenerujListeUzytkownikow()
    {
        listaKluczy =  model.listaUZ();
        Iterator<String[]> i = listaKluczy.iterator();
        String[] lista;
        if(listaKluczy.size()!=0)
        lista = new String[listaKluczy.size()];
        else lista =  new String[] { "" };
```

```
77          int j = 0;
78          while(i.hasNext())
79          {
80              lista[j] = i.next()[0];
81              j++;
82          }
83          return lista;
84      }
85
86      /** Odbiera od interfersju graficznego żądanie zamknięcia programu i zamyka program
87       * @param guii Referencja do interfejsu graficznego
88       */
89      public void zadanieZamknieciaProgramu(Gui guii)
90      {
91          if(guii == Program.komponentGui(this))
92          {
93              if(biezacaRozmowa != null)
94              {
95                  rozlacz();
96              }
97              if(online) offlineUstaw();
98              logowanie.wyloguj();
99              System.exit(0);
100         }
101
102     }
103
104     /** Odbiera od interfersju graficznego żądanie dodania użytkownika zdalnego
105      * @param param Tablica z parametrami określającymi nowego użytkownika zdalnego. param[0]:nazwa użytkownika,
    param[1]:adres ip */
106     public void dodanoUzytkownikaZdalnego(String[] param)
107     {
108         InetSocketAddress adr = new InetSocketAddress(param[1],model.zwrocAktywnyPortSluchajacy());
109         try{
110             model.dodajUzytkownikaZdalnego(param[0], adr);
111         }
112         catch(Wyjatek w)
113         {
114             odbierzWyjatek(w.getMessage());
115         }
116         odswiezListeKontaktow();
117
118     }
119
120     /** Odbiera od interfersju graficznego żądanie dodania użytkownika zdalnego
121      * @param guii Referencja do interfejsu graficznego
122      */
123     public void usunietoUzytkownikaZdalnego(Gui guii)
124     {
125         if(guii == gui)
126         if(!gui.listaUzytkownikow.isSelectionEmpty())
127         {
128             try
129             {
130                 String[] refer = zwrocUzytkownikaWolajacegoZListy();
131                 model.usunUzytkownikaZdalnego(refer[0], refer[1]);
132             } catch (Wyjatek e)
133             {
134                 odbierzWyjatek(e.getMessage());
135             }
136             odswiezListeKontaktow();
137             }
138     }
139
140     /** Zwraca użytkownika który obecnie jest zaznaczony na liście
141      * @return Tablicę z parametrami określającymi użytkownika zdalnego obecnie zaznaczonego na liście.
    param[0]:nazwa użytkownika, param[1]:adres ip
142      */
143     private String[] zwrocUzytkownikaWolajacegoZListy()
144     {
145         Iterator<String[]> i = listaKluczy.iterator();
146         int a = gui.listaOstatniIndeks;
147         for(int j = 0; j < a; j++)
148         {
149             i.next();
150         }
151         return i.next();
152     }
```

```
153
154     /** Odbiera od interfejsu graficznego żądanie przejścia w tryb online
155      * @param guii Referencja do interfejsu graficznego
156      * @param stan Tryb przejścia w online. 0:Internet, 1:Localhost1, 2:Localhost2
157      */
158     public void zmienionoNaOnline(Gui guii, int stan)
159     {
160         if(guii == Program.komponentGui(this))
161             {
162                 try {
163                     model.ustawOnline(true, LocalHost.fromInt(stan));
164
165
166                 }
167                  catch (ClassNotFoundException e) {
168                         odbierzWyjatek(e.getMessage());
169                     } catch (IOException e) {
170                         odbierzWyjatek(e.getMessage());
171                     } catch (Wyjatek e) {
172                         if(e.getMessage().contains("otworzy\u0107")) {
173                             odbierzWyjatek(e.getMessage());
174                             gui.menuOnline.setSelected(false);
175                             return;
176                         }
177                         odbierzWyjatek(e.getMessage());
178                     }
179
180                 online = true;
181                 sprawdzPrzyciskiLaczenia();
182                 switch(stan)
183                 {
184                 case 0: SwingUtilities.invokeLater(new Runnable(){public void run() { gui.napisPaska.setText("Online:
    " + Program.komponentModel(sterowanie).mojeIP()); }} ); break;
185                 case 1: SwingUtilities.invokeLater(new Runnable(){public void run() { gui.napisPaska.setText("Online:
    " + "127.0.0.1:5623"); }} ); break;
186                 case 2: SwingUtilities.invokeLater(new Runnable(){public void run() { gui.napisPaska.setText("Online:
    " + "127.0.0.1:5624"); }} ); break;
187                 }
188                 gui.dezaktywujWyborPrzelacznikowOnline();
189
190
191             }
192
193     }
194
195     /** Odbiera od interfejsu graficznego żądanie przejścia w tryb offline
196      * @param guii Referencja do interfejsu graficznego
197      */
198     public void zmienionoNaOffline(Gui guii)
199     {
200         if(guii == Program.komponentGui(this))
201             {
202             offlineUstaw();
203             online = false;
204             }
205
206     }
207
208     /** Ustawia offline w sterowaniu */
209     private void offlineUstaw()
210     {
211         if(biezacaRozmowa != null)
212             rozlacz();
213         try {
214             model.ustawOnline(false, null);
215
216         } catch (ClassNotFoundException e) {
217             odbierzWyjatek(e.getMessage());
218         } catch (IOException e) {
219             odbierzWyjatek(e.getMessage());
220         } catch (Wyjatek e) {
221             odbierzWyjatek(e.getMessage());
222         }
223         online = false;
224         sprawdzPrzyciskiLaczenia();
225         SwingUtilities.invokeLater(new Runnable(){public void run() { gui.napisPaska.setText("Offline"); }} );
226         gui.aktywujWyborPrzelacznikowOnline();
227     }
```

```
228
229     /** Odbiera od interfejsu graficznego zmianę zaznacznia na liście użytkowników
230      * @param guii Referencja do interfejsu graficznego
231      */
232     public void zmienionoStanListy(Gui guii)
233     {
234         if(guii == Program.komponentGui(this))
235         {
236             sprawdzPrzyciskiLaczenia();
237                     }
238
239
240     }
241
242     /** Odbiera od interfejsu graficznego żądanie nawiązania połączenia z użytkownikiem zdalnym
243      * @param guii Referencja do interfejsu graficznego
244      */
245     public void zadaniePolaczenia(Gui guii)
246     {
247         if(guii == gui)
248             if(!gui.listaUzytkownikow.isSelectionEmpty())
249             {
250                 pasekPostepuUstaw(true, "Łączenie...");
251                 String[] refer = zwrocUzytkownikaWolajacegoZListy();
252                 try
253                 {
254                     biezacaRozmowa = model.rozpocznijRozmowe(model.zwrocUzytkownikaZdalnego(refer[1], refer[0]));
255                 } catch (Wyjatek e)
256                 {
257                     odbierzWyjatek(e.getMessage());
258                 }
259                 finally
260                 {
261                     pasekPostepuUstaw(false,null);
262                 }
263                 pasekPostepuUstaw(false,null);
264                 if(biezacaRozmowa == null)
265                 SwingUtilities.invokeLater(new Runnable(){public void run() { gui.oknoDialogoweWyjatek("Brak
    odpowiedzi od użytkownika: " + zwrocUzytkownikaWolajacegoZListy()[0] + "(" + zwrocUzytkownikaWolajacegoZListy()[1] +
    ")");}} );
266                 else if(biezacaRozmowa.zwrocIdRozmowy() == 0)
267                 SwingUtilities.invokeLater(new Runnable(){public void run() { gui.oknoDialogoweWyjatek("Użytkownik: "
    + zwrocUzytkownikaWolajacegoZListy()[0] + "(" + zwrocUzytkownikaWolajacegoZListy()[1] + ") nie zgodził się na
    rozpoczęcie rozmowy..");}} );
268                 else
269                 {
270                 SwingUtilities.invokeLater(new Runnable(){public void run() { gui.panel1Etykieta.setText("Połączono
    z: " + zwrocUzytkownikaWolajacegoZListy()[0] + ", IP: " + zwrocUzytkownikaWolajacegoZListy()[1]);
271                                                                             gui.aktywujPanelRozmowa();}} );
272                 sprawdzPrzyciskiLaczenia();
273                 }
274             }
275
276     }
277
278     /** Odbiera od interfejsu graficznego żądanie zakończenia połączenia z użytkownikiem zdalnym
279      * @param guii Referencja do interfejsu graficznego
280      */
281     public void zadanieRozlaczenia(Gui guii)
282     {
283         if(guii == gui)
284         {
285             rozlacz();
286         }
287     }
288
289     /** Odbiera od interfejsu graficznego żądanie wysłania pliku do uczestników rozmowy
290      * @param guii Referencja do interfejsu graficznego
291      */
292     public void zadanieWyslaniaPliku(Gui guii)
293     {
294         if(guii == gui)
295         {
296         FileDialog oknoPlik = new FileDialog(gui, "Wybierz plik", FileDialog.LOAD);
297         oknoPlik.setVisible(true);
298         String sciezkaPliku = oknoPlik.getDirectory() + oknoPlik.getFile();
299         if(sciezkaPliku.isEmpty()) pasekPostepuUstaw(true, "Wysłano żądanie odebrania pliku. Oczekiwanie na
    odpowiedź...");
```

```
300         try {
301             StanWyslaniaPliku stan = model.wyslijPlik(null, biezacaRozmowa, sciezkaPliku);
302             pasekPostepuUstaw(false,null);
303             if(stan.stan == 1) SwingUtilities.invokeLater(new Runnable(){public void run() {
    gui.oknoDialogoweInfo("Plik został wysłany!", "Stan wysyłania pliku");}} );
304             else if(stan.stan == 0) SwingUtilities.invokeLater(new Runnable(){public void run() {
    gui.oknoDialogoweInfo("Użytkownik nie zgadza się na odbiór pliku!", "Stan wysyłania pliku");}} );
305             else if(stan.stan == -1) SwingUtilities.invokeLater(new Runnable(){public void run() {
    gui.oknoDialogoweInfo("Użytkownik nie odpowiada na żądanie odebrania pliku!", "Stan wysyłania pliku");}} );
306             else ;
307 
308 
309         } catch (Wyjatek e) {
310             odbierzWyjatek(e.toString());
311         }
312         }
313     }
314 
315     /** Zakańcza bieżącą rozmowę */
316     private void rozlacz()
317     {
318         try {
319             model.zakonczRozmowe(biezacaRozmowa);
320         } catch (IOException e) {
321             odbierzWyjatek(e.getMessage() + e.getCause());
322         }
323         //model.zakonczRozmowe(biezacaRozmowa);
324         biezacaRozmowa = null;
325         sprawdzPrzyciskiLaczenia();
326         gui.dezaktywujPanelRozmowa();
327         SwingUtilities.invokeLater(new Runnable(){public void run() { gui.panel1Etykieta.setText("Rozłączono");}} );
328     }
329 
330     /** Odbiera od interfejsu graficznego żądanie wysłania wiadomości do uczestników rozmowy
331      * @param guii Referencja do interfejsu graficznego
332      */
333     public void zadanieWyslaniaWiadomosci(Gui guii)
334     {
335         if(guii == gui)
336         {
337             zmiennaText = gui.panel1PoleTekstowesc.getText();
338 
339             try {
340                 model.wyslijWiadomosc(biezacaRozmowa, new String(gui.panel1PoleTekstoweWpissc.getText()));
341             } catch (IOException e) {
342             odbierzWyjatek(e.getMessage()+e.getCause());
343             }
344             SwingUtilities.invokeLater(new Runnable(){public void run() {
345             zmiennaText = zmiennaText.concat("\n" + gui.panel1PoleTekstoweWpissc.getText());
346             gui.panel1PoleTekstowesc.setText(zmiennaText);
347             gui.panel1PoleTekstoweWpissc.setText("");
348             }} );
349         }
350     }
351 
352     /** Funkcja zarządza przyciskami związanymi z połączeniem */
353     private void sprawdzPrzyciskiLaczenia()
354     {
355         if(online && !gui.listaUzytkownikow.isSelectionEmpty())
356         {
357             SwingUtilities.invokeLater(new Runnable(){public void run() {
358                                                     if(biezacaRozmowa!=null)
359                                                     {
360 
361 
    if(biezacaRozmowa.zwrocNazweUzykownika().equals(zwrocUzytkownikaWolajacegoZListy()[0]))
362                                                                 { gui.panel2Polacz.setEnabled(false);
363                                                                   gui.panel2Rozlacz.setEnabled(true); }
364 
365                                                         else
366                                                         {
367                                                         gui.panel2Polacz.setEnabled(false);
368                                                         gui.panel2Rozlacz.setEnabled(false);
369                                                         }
370                                                     }
371                                                     else
372                                                         {
373                                                             gui.panel2Polacz.setEnabled(true);
```

```
374                                                             gui.panel2Rozlacz.setEnabled(false);
375                                                         }
376                                                     }} );
377         }
378         else
379             gui.dezaktywujPrzyciskiPaneluUzytkownicyLaczenie();
380 
381     }
382 
383     //-----------------------------------------------------------------------------------------Interfejs prawy
384     @Override
385     public void odbierzWiadomosc(String ip, RozmowaID id, String wiadomosc) {
386         String text = gui.panel1PoleTekstowesc.getText();
387         Calendar czas = Calendar.getInstance();
388         gui.panel1PoleTekstowesc.setText(text.concat("\n\t" + czas.get(Calendar.HOUR_OF_DAY)
389                 + ":" + czas.get(Calendar.MINUTE) + ":" + czas.get(Calendar.SECOND) +" " + wiadomosc));
390     }
391 
392     @Override
393     public OdpowiedzSterowania odbierzRozmowe(String ip, RozmowaID rozmowa) {
394         int odp = gui.oknoDialogowePytanie("Czy zgadzasz si\u0119 na rozpocz\u0119cie rozmowy z
    nast\u0119puj\u0105cym u\u017cytkownikiem:\n"
395                 + "u\u017cytkownik " + rozmowa.zwrocNazweUzykownika() + ", IP: " + ip + ".");
396         switch(odp)
397         {
398         case -1: return new OdpowiedzSterowania(-1);
399         case 0: return new OdpowiedzSterowania(0);
400         case 1: {
401 
402             SwingUtilities.invokeLater(new Runnable(){public void run() { gui.panel1Etykieta.setText("Połączono z: "
    + zwrocUzytkownikaWolajacegoZListy()[0] + ", IP: " + zwrocUzytkownikaWolajacegoZListy()[1]);
403                                                                     gui.aktywujPanelRozmowa(); }} );
404             odswiezListeKontaktow();
405             biezacaRozmowa = rozmowa;
406             sprawdzPrzyciskiLaczenia();
407             return new OdpowiedzSterowania(1);
408         }
409         }
410         return null;
411 
412     }
413 
414     /** Odbiera zakończenie rozmowy spowodowane niespodziewanym rozłączeniem */
415     public void odbierzZakonczenieRozmowyWymuszone() {
416             model.rozmowy().get(biezacaRozmowa.zwrocIdRozmowy()).zakonczWymuszenie();
417         SwingUtilities.invokeLater(new Runnable(){public void run() {
418             gui.dezaktywujPanelRozmowa();
419             gui.panel1Etykieta.setText("Rozłączono"); }} );
420             biezacaRozmowa = null;
421             sprawdzPrzyciskiLaczenia();
422 
423     }
424 
425     @Override
426     public void odbierzZakonczenieRozmowy(String ip, RozmowaID id) {
427         SwingUtilities.invokeLater(new Runnable(){public void run() {
428             gui.dezaktywujPanelRozmowa();
429             gui.panel1Etykieta.setText("Rozłączono"); }} );
430             biezacaRozmowa = null;
431             sprawdzPrzyciskiLaczenia();
432 
433     }
434 
435     @Override
436     public void odbierzWyjatek(String opis)
437     {
438         gui.oknoDialogoweWyjatek(opis);
439 
440     }
441 
442     @Override
443     public OdpowiedzSterowania odbierzPlik(String ip, RozmowaID id, PlikNaglowek plik, StanWyslaniaPliku postep)
444     {
445         long dlug = plik.danePliku.length();
446 
447         int odp = gui.oknoDialogowePytanie("Czy zgadzasz się na odebranie następującego pliku:\n"
448                 + "u\u017cytkownik: " + id.zwrocNazweUzykownika() + ", IP: " + ip + "\n"
449                 + "nazwa pliku: " + plik.danePliku.getName() + ", rozmiar: " + dlug + " bajtów");
```

```
450         switch(odp)
451         {
452         case -1: return new OdpowiedzSterowania(-1);
453         case 0: return new OdpowiedzSterowania(0);
454         case 1: pasekPostepuUstaw(true,"Pobieranie pliku...");
455             return new OdpowiedzSterowania(1);
456         }
457         return new OdpowiedzSterowania(-1);
458     }
459 
460     /** Informuje o odebraniu wcześniej uzgodnionego pakietu z danymi (pliku)
461      * @param nazwaKoncowa Nazwa pliku
462      * @return Ścieżka pliku, gdzie plik będzie zapisany
463      */
464     public String odebranoPlik(String nazwaKoncowa)
465     {
466         pasekPostepuUstaw(false,null);
467         FileDialog oknoPlik = new FileDialog(gui, "Wybierz katalog docelowy", FileDialog.SAVE);
468         oknoPlik.setFile(nazwaKoncowa);
469         oknoPlik.setVisible(true);
470         return oknoPlik.getDirectory() + oknoPlik.getFile();
471 
472     }
473 
474     /** Zarządza paskiem postępu
475      * @param aktywny Wartość "true" sprawia że pasek jest widoczny, wartość "false" przeciwnie.
476      * @param tekst Tekst wyświetlany na pasku
477      */
478     public void pasekPostepuUstaw(final boolean aktywny, final String tekst)
479     {
480         try {
481             SwingUtilities.invokeAndWait(new Runnable(){public void run() {
482                 if(aktywny)
483                 gui.pasekPostepu.setVisible(true);
484                 else gui.pasekPostepu.setVisible(false);
485                 if(tekst != null)
486                     {
487                     gui.pasekPostepu.setString(tekst);
488                     gui.pasekPostepu.setStringPainted(true);
489                     }
490                 else gui.pasekPostepu.setStringPainted(false);
491             }});
492         } catch (InvocationTargetException e) {
493             // TODO Auto-generated catch block
494             e.printStackTrace();
495         } catch (InterruptedException e) {
496             // TODO Auto-generated catch block
497             e.printStackTrace();
498         }
499 
500 
501     }
502 
503 }
504 
505 
506 
507 
```

```java
package pl.nalazek.komunikator.logika;

import java.util.*;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa obsługująca wątki w programie*/
public class Watek extends Thread{

    /** Referencja do modelu, dla obsługi pakietów */
    protected static Model modelRef;
    /** Lista przechowująca wszystkie aktywne wątki*/
    static ArrayList<Watek> listaWatkow = null;

    /** Konstruktor */
    public Watek()
    {
        if(listaWatkow == null) { listaWatkow = new ArrayList<Watek>(); }
        listaWatkow.add(Watek.this);
    }

    /** Rozserzona metoda, oddziedziczona z klasy Thread. Dodatkowo usuwa aktualny wątek z listy wątków*/
    public void interrupt()
    {
        super.interrupt();
        listaWatkow.remove(Watek.this);
    }

    /** Zabija wątek */
    public void zabijWatek()
    {
        interrupt();
    }

    /** Zabija wszystkie stworzone wątki*/
    public void zabijWszystkieWatki()
    {
        Iterator<Watek> itertr = listaWatkow.iterator();
        while(itertr.hasNext())
        {
            itertr.next().interrupt();
        }
    }

    /** Ustawia referencję do modelu programu */
    static void ustawRefModelu(Model m)
    {
        modelRef = m;
    }

    /** Sprawdza czy dany wątek jeszcze istnieje */
    public static boolean sprawdzCzyIstniejeWatek(Watek w)
    {
        if(listaWatkow.contains(w)) return true;
        else return false;
    }
}

```

```java
package pl.nalazek.komunikator.logika;

import java.io.*;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketException;
import pl.nalazek.komunikator.Program;
import pl.nalazek.komunikator.logika.logowanie.UzytkownikZdalny;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa uruchumiająca nowy wątek do obsługi przychodzących pakietów */
public class WatekObslugujacyPakiet extends Watek{

    private ObjectInputStream strumienPrzych;
    private Socket gniazdoSluchajace = null;
    private String adresIp;

    /** Konstruktor
     * @param iS Obietkowy strumień przychodzący
     * @param gniazdoSluchajace Gniazdo do którego podpięty jest strumień
     */
    public WatekObslugujacyPakiet(ObjectInputStream iS, Socket gniazdoSluchajace)
    {
        strumienPrzych = iS;
        this.gniazdoSluchajace = gniazdoSluchajace;
    }

    /**  Główna metoda wątku */
    public void run()
    {
        try
        {
            InetSocketAddress adresSockt = (InetSocketAddress)gniazdoSluchajace.getRemoteSocketAddress();
            String nazwaUzytkownika;
            if(adresSockt.getAddress().isLoopbackAddress()) adresIp = "127.0.0.1";
            else adresIp = adresSockt.getAddress().getHostAddress();
            while(!interrupted()) {
            Pakiet pakiet = (Pakiet)strumienPrzych.readObject();
            nazwaUzytkownika = pakiet.zwrocIdRozmowyZrodlowej().zwrocNazweUzykownika();
            System.out.println("Pakiet obsłużony");
            UzytkownikZdalny uZdalny = modelRef.zwrocUzytkownikaZdalnego(adresIp, nazwaUzytkownika);


        NrIdPakietu nr = pakiet.getNrId();

        /** Pakiet z wiadomością */
        if(nr == NrIdPakietu.WIADOMOSC)
        {
        Wiadomosc wiadomosc = (Wiadomosc)pakiet;
        Program.komponentSterowanie().odbierzWiadomosc(adresIp, wiadomosc.zwrocIdRozmowyDocelowej(),
  wiadomosc.zwrocTekst());
        }

        /** Pakiet z żądaniem odebrania danych */
        else if(nr == NrIdPakietu.DANE_PYTANIE)
        {
            DanePytanie pytanie = (DanePytanie)pakiet;
            OdpowiedzSterowania kod = Program.komponentSterowanie().odbierzPlik(adresIp,
  pytanie.zwrocIdRozmowyDocelowej(), pytanie.plikInfo, null);
            switch(kod.kod)
            {
            case 0: //odmowa
                synchronized(modelRef.rozmowy().get(pytanie.zwrocIdRozmowyDocelowej().zwrocIdRozmowy()).semaforKoleje
  k)
                { modelRef.rozmowy().get(pytanie.zwrocIdRozmowyDocelowej().zwrocIdRozmowy())
                    .obslugaKolejki(uZdalny, new DaneOdpowiedz(pytanie,false)); }
                break;
            case 1: //zgoda
                modelRef.pytania().put(pytanie.zwrocIdRozmowyDocelowej().zwrocIdRozmowy(), null);
                synchronized(modelRef.rozmowy().get(pytanie.zwrocIdRozmowyDocelowej().zwrocIdRozmowy()).semaforKoleje
  k)
                {
                modelRef.rozmowy().get(pytanie.zwrocIdRozmowyDocelowej().zwrocIdRozmowy())
                    .obslugaKolejki(uZdalny, new DaneOdpowiedz(pytanie,true)); }
```

```
75
76                  try {
77
78
79                          for(int i = 0; i<800; i++)
80                          {
81                              Thread.sleep(20);
82                              /* sprawdzenie czy odpowiedz która przyszła jest opdowiedzią oczekiwaną */
83                              Dane dane = (Dane)
    modelRef.pytania().get(pytanie.zwrocIdRozmowyDocelowej().zwrocIdRozmowy());
84                              if(dane != null)
85                                  {
86                                  /* Odpowiedź pozytywna */
87                                  if(dane.zwrocKlucz() == pytanie.zwrocKlucz())
88                                          {
89                                          dane.zwrocPlik().zapiszPlik(kod.sciezka);
90                                          modelRef.pytania().remove(pytanie.zwrocKlucz());
91                                          }
92                                  /* Odpowiedź negatywna */
93                                  else
94                                          {
95                                          modelRef.pytania().remove(pytanie.zwrocKlucz());
96                                          }
97                                  }
98                          }
99                          modelRef.pytania().remove(pytanie.zwrocKlucz());
100
101                     }
102                     catch (InterruptedException e)
103                     {
104                         throw new Wyjatek("W oczekiwaniu na odpowiedź wystąpił wyjątek.\n" + e.getMessage() + ";;" +
    e.getCause());
105                     }
106                     catch (IOException e)
107                     {
108                     throw new Wyjatek("Przy zapisywaniu pliku wystąpił wyjątek.\n" + e.getMessage() + ";;" +
    e.getCause());
109                     }
110                 break;
111             case -1: //brak odpowiedzi
112                 break;
113             }
114         }
115
116         /** Pakiet z żądaniem rozmowy */
117         else if(nr == NrIdPakietu.ROZMOWA_PYTANIE)
118         {
119             RozmowaPytanie pytanie = (RozmowaPytanie)pakiet;
120             /* sprawdź użytkownika zdalnego w kontaktach*/
121             if(uZdalny == null)
122                 uZdalny = modelRef.dodajUzytkownikaZdalnego(nazwaUzytkownika, adresSockt);
123             /* stworzenie rozmowy */
124             Rozmowa rozmowa = new Rozmowa(modelRef.zwrocAktywnyUzytkownikLokalny(),uZdalny);
125             OdpowiedzSterowania kod = Program.komponentSterowanie().odbierzRozmowe(adresIp,
    rozmowa.zwrocIdRozmowy());
126             switch(kod.kod)
127             {
128             case 0: //odmowa
129                 //( new WatekWysylajacy(new RozmowaOdpowiedz(pytanie, new RozmowaID(0,"UżytkownikNegatywny")),
    adresIp, Model.zwrocAktywnyPortSluchajacyPrzeciwny()) ).start();
130                 synchronized(rozmowa.semaforKolejek)
131                 { rozmowa.obslugaKolejki(uZdalny, new RozmowaOdpowiedz(pytanie, new
    RozmowaID(0,"UżytkownikNegatywny"))); }
132                 break;
133             case 1: //zgoda
134                 rozmowa.dodajIdRozmowyDocelowej(uZdalny, pytanie.zwrocIdRozmowyZrodlowej());
135
136                 /* dodanie do aktywnych rozmów */
137                 modelRef.rozmowy().put(rozmowa.zwrocIdRozmowy().zwrocIdRozmowy(), rozmowa);
138
139                 /* wysłanie pakietu potwierdzającego */
140                 synchronized(rozmowa.semaforKolejek)
141                 { rozmowa.obslugaKolejki(uZdalny, new RozmowaOdpowiedz(pytanie,rozmowa.zwrocIdRozmowy())); }
142                 break;
143             case -1: //brak odpowiedzi
144                 break;
145             }
146
```

```
147         }
148
149         /** Pakiet z opdowiedzią na żądanie rozmowy */
150         else if(nr == NrIdPakietu.ROZMOWA_ODPOWIEDZ)
151         {
152         RozmowaOdpowiedz odpowiedz = (RozmowaOdpowiedz)pakiet;
153         if( modelRef.pytania().containsKey(odpowiedz.zwrocIdRozmowyDocelowej().zwrocIdRozmowy()) )
154             modelRef.pytania().put(odpowiedz.zwrocIdRozmowyDocelowej().zwrocIdRozmowy(), odpowiedz);
155         else throw new Wyjatek("Odebrano przeterminowany  pakiet z odpowiedzią na żądanie rozmowy!");
156         }
157
158
159         /** Pakiet z opdowiedzią na żądanie danych */
160         else if(nr == NrIdPakietu.DANE_ODPOWIEDZ)
161         {
162         DaneOdpowiedz odpowiedz = (DaneOdpowiedz)pakiet;
163         if( modelRef.pytania().containsKey(odpowiedz.zwrocIdRozmowyDocelowej().zwrocIdRozmowy()) )
164             modelRef.pytania().put(odpowiedz.zwrocIdRozmowyDocelowej().zwrocIdRozmowy(), odpowiedz);
165         else throw new Wyjatek("Odebrano przeterminowany pakiet z odpowiedzią na żądanie danych!");
166         }
167
168         /** Pakiet z danymi */
169         else if(nr == NrIdPakietu.DANE)
170         {
171         Dane dane = (Dane)pakiet;
172         if( modelRef.pytania().containsKey(dane.zwrocIdRozmowyDocelowej().zwrocIdRozmowy()) )
173             {modelRef.pytania().remove(dane.zwrocIdRozmowyDocelowej().zwrocIdRozmowy());
174         String sciezka = Program.komponentSterowanie().odebranoPlik(dane.plikInfo.danePliku.getName());
175         dane.zwrocPlik().zapiszPlik(sciezka);
176             }
177         else throw new Wyjatek("Odebrano pakiet z danymi! Pakiet potwierdzający nie był wysyłany...");
178         }
179
180         /** Rozmowa koniec */
181         else if(nr == NrIdPakietu.ROZMOWA_KONIEC)
182         {
183         RozmowaKoniec pakietKoniec = (RozmowaKoniec)pakiet;
184         if( modelRef.rozmowy().containsKey(pakietKoniec.zwrocIdRozmowyDocelowej().zwrocIdRozmowy()))
185             {
186             modelRef.rozmowy().get(pakietKoniec.zwrocIdRozmowyDocelowej().zwrocIdRozmowy()).usunUzytkownikaZdalnego(m
    odelRef.zwrocUzytkownikaZdalnego(adresIp, nazwaUzytkownika), false);
187             Program.komponentSterowanie().odbierzZakonczenieRozmowy(adresIp, pakietKoniec.zwrocIdRozmowyDocelowej());
188             }
189
190         }
191
192         /** Inny pakiet */
193         else if(nr == NrIdPakietu.PAKIET)
194         {
195
196         }
197
198         /** Nieznany pakiet */
199         else
200         {
201             throw new Wyjatek("Odebrano nieznany pakiet!");
202         }
203         }
204         }
205         catch(ClassNotFoundException w)
206         {
207             Program.komponentSterowanie().odbierzWyjatek("W wątku: " + this.getName() + " wystąpił wyjątek.\n" +
    w.getMessage() + ";;" + w.getCause());
208             System.out.println(w.getMessage() + " " + w.getCause());
209         }
210         catch(EOFException w)
211         {
212             Program.komponentSterowanie().odbierzWyjatek("Zakończono połączenie przychodzące z " + adresIp + "!");
213         }
214         catch(SocketException w)
215         {
216             if(w.getMessage().contains("Connection reset")) {
217                 Program.komponentSterowanie().odbierzWyjatek("Zakończono połączenie przychodzące z " + adresIp +
    "!");
218                 Program.komponentSterowanie().odbierzZakonczenieRozmowyWymuszone();
219             }
220         }
221
```

```
222         catch(IOException w)
223         {
224             Program.komponentSterowanie().odbierzWyjatek("W wątku: " + this.getName() + " wystąpił wyjątek.\n" +
    w.getMessage() + ";;" + w.getCause());
225             System.out.println(w.getMessage() + " " + w.getCause());
226         }
227         catch(Exception w)
228         {
229             Program.komponentSterowanie().odbierzWyjatek("W wątku: " + this.getName() + " wystąpił wyjątek.\n" +
    w.getMessage() + ";;" + w.getCause());
230             System.out.println(w.getMessage() + " " + w.getCause());
231         }
232 
233 
234     }
235 
236 
237 }
238 
```

```java
package pl.nalazek.komunikator.logika;

import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InterruptedIOException;
import java.io.ObjectInputStream;
import java.net.ServerSocket;
import java.net.Socket;
import javax.swing.ProgressMonitorInputStream;
import pl.nalazek.komunikator.Program;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Wątek obsługujący nowe połączenia przychodzące */
public class WatekSluchajacy extends Watek {

    private ServerSocket gniazdoSluchajaceSrw = null;
    private int port;
    private Socket gniazdoSluchajace = null;
    private ProgressMonitorInputStream pmis;
    private BufferedInputStream bis;
    private ObjectInputStream strumienPrzych = null;

    /** Konstruktor domyślny */
    public WatekSluchajacy()
    { }

    /** Konstruktor z parametrem
     * @param port Numer portu, który ma zostać otwarty do nasłuchiwania
     * @throws IOException
     * @throws ClassNotFoundException
     * @throws Wyjatek
     */
    public WatekSluchajacy(int port) throws IOException, ClassNotFoundException, Wyjatek
    {
        this.port = port;
    }

    /**  Główna metoda wątku */
    public void run()
    {
        try{

            gniazdoSluchajaceSrw = new ServerSocket(port);
            gniazdoSluchajaceSrw.setSoTimeout(1000);
            gniazdoSluchajace = new Socket();
        while(!interrupted())
            {
            try{
            gniazdoSluchajace = /*(SSLSocket)*/gniazdoSluchajaceSrw.accept();
            pmis = new ProgressMonitorInputStream(Program.komponentGui(this),"Pobieranie
  pliku...",gniazdoSluchajace.getInputStream());
            bis= new BufferedInputStream(pmis);
            strumienPrzych = new ObjectInputStream(bis);
            WatekObslugujacyPakiet a = new WatekObslugujacyPakiet(strumienPrzych, gniazdoSluchajace);
            a.start();
            }
            catch(InterruptedIOException w) {};
            }
        if(strumienPrzych!=null){
            strumienPrzych.close();
            bis.close();
            pmis.close();
        }

        gniazdoSluchajace.close();
        gniazdoSluchajaceSrw.close();

        }
        catch(IOException w)
        {
            System.out.println(w.getMessage());
        }
        finally
        {
```

```
78              try {
79                  if(gniazdoSluchajace != null) gniazdoSluchajace.close();
80                  if(gniazdoSluchajaceSrw!=null) gniazdoSluchajaceSrw.close();
81                  if(strumienPrzych!=null){
82                      strumienPrzych.close();
83                      bis.close();
84                      pmis.close();
85                  }
86              } catch (IOException e) {
87              }
88          }
89      }
90  }
91
```

```java
package pl.nalazek.komunikator.logika;

import java.io.*;
import java.net.Socket;
import pl.nalazek.komunikator.Program;
import pl.nalazek.komunikator.logika.logowanie.UzytkownikZdalny;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Watek tworzący połączenie do wysylania na określonym porcie */
public class WatekWysylajacy extends Watek {

    private Socket gniazdoWysylajace = null;
    private int port;
    private ObjectOutputStream strumienWych;
    private Rozmowa rozmowa;
    private UzytkownikZdalny uZdalny;
    private Pakiet pakiet;
    private String adresIp;
    /** Semafor obsługujący dostęp do strumienia wysyłającego. Umożliwia użycie na nim metod wait() i notify(). */
    Object semafor;

    /** Konstruktor wątku wysyłającego dla użytkownika zdalnego
     * @param rozmowa Referencja do rozmowy, z której będzie obsługiwany ten wątek
     * @param uZdalny Referencja do użytkownika zdalnego, który ma zostać podłączony strumień wychodzący w tym wątku
     * @param port Port użytkownika zdalnego do którego ma zostać podłączony strumień wychodzący w tym wątku
     */
    public WatekWysylajacy(Rozmowa rozmowa, UzytkownikZdalny uZdalny, int port)
    {
        semafor = new Object();
        this.port = port;
        this.uZdalny = uZdalny;
        this.rozmowa = rozmowa;
    }

    /** Konstruktor wątku wysyłającego dla pojedynczego pakietu
     * @param pakiet Pakiet do wysłania
     * @param adresIp Adres IP komputera docelowego
     * @param port Port komputera docelowego
     */
    public WatekWysylajacy(Pakiet pakiet, String adresIp, int port)
    {
        semafor = new Object();
        this.port = port;
        this.adresIp = adresIp;
        this.pakiet = pakiet;
    }

    /**  Główna metoda wątku */
    public void run()
    {
        try{

            if(uZdalny != null)
            {
                String ip = (uZdalny.zwrocIp().equals("localhost"))
                        || (uZdalny.zwrocIp().equals("127.0.0.1")) ?
                        null : uZdalny.zwrocIp();
                gniazdoWysylajace = new Socket(ip , port);
            }
            else if(adresIp != null)
                gniazdoWysylajace = new Socket(adresIp=="localhost" ? null : adresIp, port);
            else
                interrupt();
            BufferedOutputStream bos = new BufferedOutputStream(gniazdoWysylajace.getOutputStream());
            strumienWych = new ObjectOutputStream(bos);
            if(uZdalny != null)
            {
                synchronized(semafor) {

                    while(!interrupted())
                    {

                        semafor.wait();

```

```
79                          Pakiet a =  rozmowa.obslugaKolejki(uZdalny,null);
80                          strumienWych.writeObject(a);
81                          strumienWych.flush();
82                          strumienWych.reset();
83                      }
84                  }
85              }
86              else if(adresIp != null)
87              {
88                  strumienWych.writeObject(pakiet);
89                  strumienWych.flush();
90              }
91              gniazdoWysylajace.close();
92              strumienWych.close();
93          }
94          catch(IOException w)
95          {
96              if(w.getMessage().contains("refused")) ;
97              else Program.komponentSterowanie().odbierzWyjatek("W wątku: " + this.getName() + " wystąpił wyjątek.\n" +
    w.getMessage() + ";;" + w.getCause());
98          }
99          catch(InterruptedException w)
100         {
101         }
102         catch(Throwable w)
103         {
104             w.getMessage();
105         }
106         finally
107         {
108             try
109             {
110                 if(gniazdoWysylajace != null) gniazdoWysylajace.close() ;
111                 if(strumienWych != null) strumienWych.close();
112             }
113             catch (IOException i)
114             {
115                 i.getMessage();
116                 }
117         }
118 
119     }
120 }
121 
```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca pakiet z wiadomością */
public class Wiadomosc extends Pakiet {

    private static final long serialVersionUID = 1L;
    private String tresc = "";

    /** Konstruktor wiadomości
     * @param tresc Treść przesyłanej wiadomości
     * @param rozmowaZrodlowa Rozmowa źródłowa z której pochodzi wiadomość
     * */
    public Wiadomosc(String tresc, RozmowaID rozmowaZrodlowa)
    {
        this.tresc = tresc;
        nazwa = "Pakiet Wiadomosc";
        nrid = NrIdPakietu.WIADOMOSC;
        this.rozmowaZrodlowa = rozmowaZrodlowa;

    }
    /** Zwaraca tekst wiadomości */
    public String zwrocTekst()
    {
        return tresc;
    }

}

```

```java
package pl.nalazek.komunikator.logika;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

public class Wyjatek extends Exception {

    private static final long serialVersionUID = 1L;
    public Wyjatek(String komunikat)
    {
        super(komunikat);
    }
}

```

```java
package pl.nalazek.komunikator.logika.logowanie;

import java.io.*;

import pl.nalazek.komunikator.Program;
import pl.nalazek.komunikator.logika.Sterowanie;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa odpowiadająca za komponent Logowanie */
public class Logowanie {
    private File plik;
    private static Uzytkownicy uzytkownicy;
    private static UzytkownikLokalny zalogowanyUzytkownikLokalny;
    private Sterowanie s;

    /** Konstuktor
     * @param s Referencja do Sterowania w programie
     */
    public Logowanie(Sterowanie s)
    {
        this.s = s;
        plik = new File("uzytkownicy.dat");
        if(plik.exists())
        {
            try
            {
                ObjectInputStream ois = new ObjectInputStream(new FileInputStream(plik));
                uzytkownicy = (Uzytkownicy) ois.readObject();
                ois.close();
                uzytkownicy.wczytajUzytkownikow();
            }
            catch(Exception e)
            {
                Program.komponentSterowanie().odbierzWyjatek(e.getMessage());
            }
        }
        else
        {
            uzytkownicy = new Uzytkownicy();
        }
    }

    /** Funkcja weryfikuje poprawność danych użytkownika lokalnego i dokonuje logowania
     * @param nazwa Nazwa użytkownika lokalnego
     * @param haslo Hasło użytkownika lokalnego
     * @return UżytkownikLokalny w przypadku poprawnej weryfikacji, w przeciwnym razie "null"
     */
    public UzytkownikLokalny zaloguj(String nazwa, String haslo)
    {
        zalogowanyUzytkownikLokalny = uzytkownicy.zwrocUzytkownikaLokalnego(haslo+"/"+nazwa);
        if(zalogowanyUzytkownikLokalny != null)
        {
            Program.komponentModel(s).zalogowano(zalogowanyUzytkownikLokalny,
zalogowanyUzytkownikLokalny.zwrocListeKontaktow());
            return zalogowanyUzytkownikLokalny;
        }
        else return null;
    }

    /** Funkcja wylogowywuje obecnie zalogowanego użytkownika lokalnego */
    public void wyloguj()
    {
        if(zalogowanyUzytkownikLokalny != null)
        {
            zalogowanyUzytkownikLokalny = null;
            Program.komponentModel(s).wylogowano();
            zapiszStan();
        }
        else zapiszStan();
    }

    /** Funkcja zapisuje dane logowania, w tym również kontakty na dysk */
    private void zapiszStan()
    {
```

```
78          uzytkownicy.zapiszUzytkownikow();
79          try
80          {
81              ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(plik));
82              oos.writeObject(uzytkownicy);
83              oos.close();
84          }
85          catch(Exception e)
86          {
87              Program.komponentSterowanie().odbierzWyjatek(e.getMessage());
88          }
89      }
90
91
92  }
93
```

```java
package pl.nalazek.komunikator.logika.logowanie;

import java.io.Serializable;
import java.util.HashMap;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca użytkowników w programie */
public class Uzytkownicy implements Serializable {

    private static final long serialVersionUID = 1L;
    protected int licznik;
    protected Integer nrUzytkownika = 0;
    protected String nazwa;
    static protected volatile HashMap<String,UzytkownikLokalny> uzytkownicyHasla;
    private HashMap<String,UzytkownikLokalny> uzytkownicyHaslaSer;

    /** Konstruktor */
    public Uzytkownicy()
    {
        if(uzytkownicyHasla == null)
            {
            uzytkownicyHasla = new HashMap<String,UzytkownikLokalny>();
            uzytkownicyHasla.put("domyslny/domyslny",UzytkownikDomyslny.zwrocReferencje());
            }
        licznik++;
    }

    /** Zwraca uzytkownika lokalnego na podstawie poprawnie wpisanych danych weryfikacyjnych.
     * @param hasloUzytkownik Parametry wpisane w postaci "hasło/użytkownik". Parametr domyslny/domyslny zwraca użytkownika domyslnego
     * @return Uzytkownika lokalnego w przypadku poprawnej weryfikacji, null w przypadku niepoprawnej
     */
    UzytkownikLokalny zwrocUzytkownikaLokalnego(String hasloUzytkownik)
    {
        return uzytkownicyHasla.get(hasloUzytkownik);
    }

    /** Zwraca numer użytkownika w systemie */
    public Integer zwrocNrUzytkownika()
    {
        return nrUzytkownika;
    }

    /** Zapisuje statyczną tablicę użytkowników lokalnych do zmiennej niestatycznej */
    void zapiszUzytkownikow()
    {
        uzytkownicyHaslaSer = uzytkownicyHasla;
    }

    /** Odczytuje niestatyczną tablicę użytkowników lokalnych i wczytuje ją do zmiennej statycznej */
    void wczytajUzytkownikow()
    {
        uzytkownicyHasla = uzytkownicyHaslaSer;
    }
}

```

```java
package pl.nalazek.komunikator.logika.logowanie;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca użytkownika lokalnego domyślnego.
 * Używany gdy nie istnieje żaden inny profilowany użytkownik programu
 * Klasa jest typu singleton. Tylko jeden użytkownik domyślny może istnieć w programie.
 */
public class UzytkownikDomyslny extends UzytkownikLokalny {
    private static final long serialVersionUID = 1L;
    private static UzytkownikDomyslny referencja;

    private UzytkownikDomyslny()
    {
        nrUzytkownika = 0;
        nazwa = "Uzytkownik domyslny";
    }

    /** Metoda zwracająca referencję do obiektu
     * @return Referencję do użytkownika lokalnego
     */
    public static UzytkownikDomyslny zwrocReferencje()
    {
        if(referencja == null)
            referencja = new UzytkownikDomyslny();
        return referencja;
    }

}

```

```java
package pl.nalazek.komunikator.logika.logowanie;

import java.util.LinkedHashSet;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

/** Klasa reprezentująca użytkownika lokalnego */
public class UzytkownikLokalny extends Uzytkownicy {

    private static final long serialVersionUID = 1L;
    private LinkedHashSet<UzytkownikZdalny> kontakty;

    protected UzytkownikLokalny()
    {
        nrUzytkownika = licznik;
        kontakty = new LinkedHashSet<UzytkownikZdalny>();
    }

    /** Konstruktor tworzący nowego użytkownika lokalnego
     * @param nazwa Nazwa użytkownika lokalnego
     * @param haslo Hasło użytkownika lokalnego
     */
    public UzytkownikLokalny(String nazwa, String haslo)
    {
        this();
        this.nazwa = nazwa;
        uzytkownicyHasla.put(haslo+"/"+nazwa, this);
    }


    /** Zwraca nazwę użytkownika
     * @return Nazwa użytkownika
     */
    public String zwrocNazwe()
    {
        return nazwa;
    }

    /** Zwraca kopię listy kontaktów
     * @return Lista kontaktów w postaci zbioru klas UzytkownikZdalny
     */
    LinkedHashSet<UzytkownikZdalny> zwrocListeKontaktow()
    {
        return kontakty;
    }
}

```

```java
package pl.nalazek.komunikator.logika.logowanie;

import java.net.InetSocketAddress;

/**
 * @author Daniel Nalazek
 * Copyright (C) 2014 Daniel Nalazek
 */

public class UzytkownikZdalny extends Uzytkownicy  {

    private static final long serialVersionUID = 1L;
    private String ip;
    private InetSocketAddress adresSocket;
    private int port;

    /** Konstruktor */
    public UzytkownikZdalny()
    {
        nrUzytkownika = licznik;
    }

    /** Konstruktor uzytkownika zdalnego
     * @param adresSocket Gniazdo słuchające użytkownika zdalnego
     * @param nazwa Nazwa użytkownika zdalnego
     */
    public UzytkownikZdalny(InetSocketAddress adresSocket, String nazwa)
    {
        this();
        this.adresSocket = adresSocket;
        this.ip = adresSocket.getAddress().getHostAddress();
        port = adresSocket.getPort();
        this.nazwa = nazwa;
    }

    /** Zwraca nr ip użytkownika zdalnego
     * @return Numer ip
     */
    public String zwrocIp()
    {
        if(ip != null) return ip;
        else return null;
    }

    /** Zwraca nazwę użytkownika zdalnego
     * @return Nazwa użytkownika zdalnego
     */
    public String zwrocNazwe()
    {
        return nazwa;
    }

    /** Zwraca port słuchający użytkownika zdalnego
     * @return Numer portu
     */
    public int zwrocPort()
    {
        return port;
    }

    /** Ustawia port słuchający użytkownika zdalnego
     * @param port Numer portu
     */
    public void ustawPort(int port)
    {
        this.port = port;
    }

    /** Zmienia nazwę użytkownika zdalnego
     * @param nowaNazwa Nowa nazwa użytkownika zdalnego
     */
    public void zmienNazwe(String nowaNazwa)
    {
        nazwa = nowaNazwa;
    }

    /** Zwraca gniazdo słuchające użytkownika zdalnego
     * @return Gniazdo
```

```
79      */
80     public InetSocketAddress zwrocAdresGniazda()
81     {
82         return adresSocket;
83     }
84 }
85
```