```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 }
```