Systemy Rozproszone - Ćwiczenie 5

1 Sockety

1.1 Serwer tekstowy

Poniżej znajduje się kod serwera, ktory odbiera od klienta tekst i w odpowiedzi odsyła ten sam tekst pisany wielkimi literami. Konstruktor klasy `SocketServer` tworzy gniazdo (klasa `ServerSocket`) nasłuchujące na określonym przez użytkownika porcie. Następnie, serwer rozpoczyna obsługę żądań w metodzie `SocketServer.listen()`. Metoda ta obsługuje żądania od klientów w pętli while, w której to serwer blokuje się do czasu połączenia z klienem (metoda `ServerSocket.accept()`), a następnie po uzyskaniu połączenia obsługa żądania jest przekazywana do metody `serviceClient()`. Zauważ, że wysyłanie i odbieranie komunikatów odbywa się poprzez operowanie na strumieniach.

```
import java.io.*;
import java.net.*;

public class SocketServer {
  ServerSocket clientConn;
  //ObjectOutputStream out;
  //ObjectInputStream in;

  public SocketServer(int port) {
    System.out.println("Server connecting to port "+port)
        ;
    try {
      clientConn = new ServerSocket(port);
    }
    catch (Exception e) {
      System.out.println("Exception:" + e);
      System.exit(1);
    }
  }

  public static void main(String[] args) {
    int port = 3000;
    if (args.length > 0) {
```

```
      try {
        port = Integer.parseInt(args[0]);
      }
      catch (Exception e) {
        port = 3000;
      }
    }
    SocketServer server = new SocketServer(port);
    System.out.println("Server running on port "+port);
    server.listen();
  }

  public void listen() {
    try {
      System.out.println("Waiting for clients...");
      while (true) {
        Socket clientReq = clientConn.accept();
        System.out.println("Connection from "
          + clientReq.getInetAddress().getHostName());
        serviceClient(clientReq);
      }
    }
    catch (IOException e) {
      System.out.println("Exception:" + e);
    }
  }

  public void serviceClient(Socket s) {
    DataInputStream inStream = null;
    DataOutputStream outStream = null;
    String message;
    try {
      inStream = new DataInputStream(s.getInputStream());
      outStream = new DataOutputStream(s.getOutputStream
          ());
      message = inStream.readUTF();
      outStream.writeUTF(message.toUpperCase());

    }
    catch (IOException e) {
      System.out.println("I/O Exception:" + e);

    }
  }
}
```

1.2 Klient tekstowy

W celu połączenia klienta z serwerem, klient musi znać punkt końcowy serwera (adres i port). Parametry te możesz przekazać do wykonania programu ustawiając we właściwościach projektu pole Run→Arguments na wartość "localhost 3000" (w celu połączenia z uprzednio uruchomionym serwerem lokalnym). Klient w konstruktorze tworzy gniazdo do serwerwa (obiekt klasy `Socket`), a następnie rozpoczyna komunikację z serwerem wywołując metodę `SocketClient.sendMessage()`.

```
import java.io.*;
import java.net.*;

public class SocketClient {
  Socket serverConn;

  public SocketClient(String host, int port) {
    try {
      serverConn = new Socket(host, port);
    }
    catch (Exception e){
      System.out.println("Exception:" + e);
      System.exit(1);
    }
    System.out.format("Connected to %s:%d", host, port);
  }

  public void sendMessage() {
    DataInputStream inStream = null;
    DataOutputStream outStream = null;

    String message = "hello world";
    String response;

    try {
      inStream = new DataInputStream(serverConn.
          getInputStream());
      outStream = new DataOutputStream(serverConn.
          getOutputStream());

      outStream.writeUTF(message);
      response = inStream.readUTF();

      System.out.println("Server returned " + response);
    }
    catch (IOException e) {
      System.out.println("I/O Exception:" + e);
```

```
    }

  }

  public static void main(String[] args) {
    if (args.length < 2) {
      System.out.println("Usage: java SocketClient host
          port");
      System.exit(1);
    }
    String host = args[0];
    int port;
    try {
      port = Integer.parseInt(args[1]);
    }
    catch (NumberFormatException e) {
      port = 3000;
    }

    SocketClient client = new SocketClient(host, port);
    client.sendMessage();

  }

}
```

1.3 Serwer obiektowy

Ten serwer różni si od poprzedniego sposobem wymiany komunikatów. W tym przypadku klient przekazuje serwerowi obiekt (tablica typu int) i odsyła klientowi liczbę całkowitą (równą sumie elementów w tablicy). Uwaga: kolejność w jakiej tworzone są strumienie `ObjectInputStream` i `ObjectOutputStream` jest istotna. Najpierw należy utworzyć obiekt `ObjectOutputStream`, a następnie obiekt `ObjectInputStream`.

```
import java.io.*;
import java.net.*;

public class ObjectServer {
  ServerSocket clientConn;
  //ObjectOutputStream out;
  //ObjectInputStream in;


  public ObjectServer(int port) {
```

```
    System.out.println("Server connecting to port "+port)
        ;
    try {
      clientConn = new ServerSocket(port);
    }
    catch (Exception e) {
      System.out.println("Exception:" + e);
      System.exit(1);
    }
  }


  public static void main(String[] args) {
    int port = 3000;
    if (args.length > 0) {
      try {
        port = Integer.parseInt(args[0]);
      }
      catch (Exception e) {
        port = 3000;
      }
    }
    ObjectServer server = new ObjectServer(port);
    System.out.println("Server running on port "+port);
    server.listen();
  }


  public void listen() {
    try {
      System.out.println("Waiting for clients...");
      while (true) {
        Socket clientReq = clientConn.accept();
        System.out.println("Connection from "
          + clientReq.getInetAddress().getHostName());
        serviceClient(clientReq);
      }
    }
    catch (IOException e) {
      System.out.println("Exception:" + e);
    }
  }

  public void serviceClient(Socket s) {
    ObjectInputStream inStream = null;
    ObjectOutputStream outStream = null;
```

```
      int[] data = null;
      int sum = 0;
      try {
        // z jakiegos powodu utworzenie inStream,
        // a nastepnie outStream powoduje
        // zablokowanie programu
        // wiecej ciekawostek na:
        // http://www.seasite.niu.edu/cs580java/
            Object_Serialization.html
        outStream = new ObjectOutputStream(s.
            getOutputStream());
        inStream = new ObjectInputStream(s.getInputStream()
            );
        try {
          data = (int [])inStream.readObject();
          for (int i=0; i<data.length; i++) {
              sum += data[i];
          }
          outStream.writeInt(sum);
          // wymuszamy zapis do strumienia
          outStream.flush();
        }
        catch (Exception e) {
            e.printStackTrace();
        }

      }
      catch (IOException e) {
        System.out.println("I/O Exception:" + e);

      }
    }
}
```

1.4 Klient obiektowy

Klient obiektowy działa w podobny sposób do klienta kekstowego. Tablica wysyłana jest na serwer w dwóch krokach: poprzez umieszczenie obiektu w strumieniu (`ObjectOutputStream.writeObject()`), oraz faktycznie wysłanie danych (metoda `ObjectOutputStream.flush()`).

```
import java.io.*;
import java.net.*;

public class ObjectClient {
  Socket serverConn;
```

```
public ObjectClient(String host, int port) {
  try {
    serverConn = new Socket(host, port);
  }
  catch (Exception e){
    System.out.println("Exception:" + e);
    System.exit(1);
  }
  System.out.format("Connected to %s:%d", host, port);
}

public void sendMessage() {
  ObjectInputStream inStream = null;
  ObjectOutputStream outStream = null;

  int[] data = new int[10];
  for (int i=0; i<10; i++) {
      data[i] = i+1;
  }
  int sum;

  try {
    outStream = new ObjectOutputStream(serverConn.
        getOutputStream());
    inStream = new ObjectInputStream(serverConn.
        getInputStream());

    outStream.writeObject(data);
    outStream.flush();

    sum = inStream.readInt();

    System.out.println("Server returned sum=" + sum);
  }
  catch (IOException e) {
    System.out.println("I/O Exception:" + e);
  }

}

public static void main(String[] args) {
  if (args.length < 2) {
    System.out.println("Usage: java SocketClient host
        port");
    System.exit(1);
```

```
    }
    String host = args[0];
    int port;
    try {
      port = Integer.parseInt(args[1]);
    }
    catch (NumberFormatException e) {
      port = 3000;
    }

    ObjectClient client = new ObjectClient(host, port);
    client.sendMessage();

  }

}
```

2 Przykad 1

... w którym klient otwiera połączenie z serwerem tylko na czas wykonania polecenia wybranego z menu przez użytkownika. Połączenie z serwerem zostaje utworzone po wybraniu polecenia, a po odebrania odpowiedzi od serwera połączenie jest zamykane.

2.1 Serwer

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package javaapplication1;

import java.io.*;
import java.net.*;
import java.util.*;

public class ObjectServer {
  ServerSocket clientConn;
  //ObjectOutputStream out;
  //ObjectInputStream in;


  public ObjectServer(int port) {
    System.out.println("Server connecting to port "+port)
        ;
```

```
    try {
      clientConn = new ServerSocket(port);
    }
    catch (Exception e) {
      System.out.println("Exception:" + e);
      System.exit(1);
    }
  }


  public static void main(String[] args) {
    int port = 3000;
    if (args.length > 0) {
      try {
        port = Integer.parseInt(args[0]);
      }
      catch (Exception e) {
        port = 3000;
      }
    }
    ObjectServer server = new ObjectServer(port);
    System.out.println("Server running on port "+port);
    server.listen();
  }


  public void listen() {
    try {
      System.out.println("Waiting for clients...");
      while (true) {
        Socket clientReq = clientConn.accept();
        System.out.println("Connection from "
          + clientReq.getInetAddress().getHostName());
        serviceClient(clientReq);
      }
    }
    catch (IOException e) {
      System.out.println("Exception:" + e);
    }
  }

  public void serviceClient(Socket s) {
    System.out.println("New request from " + s.
        getInetAddress());
    ObjectInputStream inStream = null;
    ObjectOutputStream outStream = null;
```

```
    int message_id;
    Object message = null;
    try {
      outStream = new ObjectOutputStream(s.
          getOutputStream());
      inStream = new ObjectInputStream(s.getInputStream()
          );
      message_id = inStream.readInt();
      System.out.println("Got message "+message_id);
      message = inStream.readObject();

      switch (message_id) {
           case 1:
                Date d = new Date();
                outStream.writeObject(d);
                break;
           case 2:
                int[] data = (int []) message;
                int sum = 0;
                for (int i=0; i<data.length; i++) {
                  sum += data[i];
                }
                outStream.writeObject(new Integer(sum));
                break;
           case 3:
                String str = (String) message;
                outStream.writeObject(str.toUpperCase());
                break;
      }
      outStream.flush();
    }
    catch (Exception e) {
      System.out.println("Exception:" + e);
    }
    System.out.println("Done.");
  }
}
```

2.2 Klient

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package javaapplication1;
```

```
import java.io.*;
import java.net.*;
import java.util.*;

public class ObjectClient {
  String host;
  int port;

  public ObjectClient(String host, int port) {
    this.host = host;
    this.port = port;
  }

  public Object sendMessage(int message_id, Object
      message) throws Exception {
    Socket serverConn;
    ObjectInputStream inStream = null;
    ObjectOutputStream outStream = null;
    Object response = null;
      serverConn = new Socket(host, port);
      outStream = new ObjectOutputStream(serverConn.
          getOutputStream());
      inStream = new ObjectInputStream(serverConn.
          getInputStream());
      outStream.writeInt(message_id);
      outStream.writeObject(message);
      outStream.flush();
      response = inStream.readObject();
      serverConn.close();
    return response;
  }

  public static void main(String[] args) {

    ObjectClient client = new ObjectClient("localhost",
        3000);

    Scanner scan = new Scanner(System.in);

    boolean end_loop = false;
    try {
      while(!end_loop) {
        System.out.println("\n\n1. Ktora godzina?");
        System.out.println("2. Suma liczb w tablicy");
        System.out.println("3. Modyfikacja tekstu");
        System.out.println("0. Koniec");
```

```
            char c = scan.nextLine().charAt(0);
            switch(c) {
                case '1':
                    Date d = (Date)client.sendMessage(1, null
                        );
                    System.out.println("Server: "+d);
                    break;
                case '2':
                    int[] data = new int[10];
                    for (int i=0; i<10; i++) {
                        data[i] = i+1;
                    }
                    Integer response = (Integer)client.
                        sendMessage(2, data);
                    System.out.println("Server: "+response);
                    break;
                case '3':
                    System.out.print("Wpisz tekst: ");
                    String str = scan.nextLine();
                    String r = (String)client.sendMessage(3,
                        str);
                    System.out.println("Server: "+r);
                    break;
                case '0':
                    end_loop = true;
                    break;
            }
          }
        }
        catch (Exception e) {
          System.out.println("Exception:" + e);
        }

    }

}
```

3 Przykad 2

... w którym klient otwiera połączenie z serwerem i utrzymuje je podczas wykonywania poleceń, do momentu jawnego zakończenia pracy przez klienta. Cała komunikacja przebiega w ramach 1 połączenia, które rozpoczynane jest przy uruchomieniu klienta.

3.1 Serwer

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package javaapplication2;

import java.io.*;
import java.net.*;
import java.util.*;

public class ObjectServer {

    ServerSocket clientConn;
    //ObjectOutputStream out;
    //ObjectInputStream in;

    public ObjectServer(int port) {
        System.out.println("Server connecting to port " +
             port);
        try {
            clientConn = new ServerSocket(port);
        } catch (Exception e) {
            System.out.println("Exception:" + e);
            System.exit(1);
        }
    }

    public static void main(String[] args) {
        int port = 3000;
        if (args.length > 0) {
            try {
                port = Integer.parseInt(args[0]);
            } catch (Exception e) {
                port = 3000;
            }
        }
        ObjectServer server = new ObjectServer(port);
        System.out.println("Server running on port " +
            port);
        server.listen();
    }

    public void listen() {
        try {
            System.out.println("Waiting for clients...");
            while (true) {
```

```
            Socket clientReq = clientConn.accept();
            System.out.println("Connection from "
                    + clientReq.getInetAddress().
                        getHostName());
            serviceClient(clientReq);
        }
    } catch (IOException e) {
        System.out.println("Exception:" + e);
    }
}

public void serviceClient(Socket s) {
    ObjectOutputStream outStream;
    ObjectInputStream inStream;
    try {
        outStream = new ObjectOutputStream(s.
            getOutputStream());
        inStream = new ObjectInputStream(s.
            getInputStream());
        int message_id;
        Object message = null;
        boolean finished = false;

        while (!finished) {
            message_id = inStream.readInt();
            System.out.println("Got message " +
                message_id);
            message = inStream.readObject();

            switch (message_id) {
                case 1:
                    Date d = new Date();
                    outStream.writeObject(d);
                    outStream.flush();
                    break;
                case 2:
                    int[] data = (int[]) message;
                    int sum = 0;
                    for (int i = 0; i < data.length;
                        i++) {
                        sum += data[i];
                    }
                    outStream.writeObject(new Integer
                        (sum));
                    outStream.flush();
                    break;
```

```
                    case 3:
                        String str = (String) message;
                        outStream.writeObject(str.
                            toUpperCase());
                        outStream.flush();
                        break;
                    case 0:
                        outStream.writeObject("bye");
                        finished = true;
                        break;
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("Done.");
    }
}
```

3.2 Klient

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package javaapplication2;

import java.io.*;
import java.net.*;
import java.util.*;

public class ObjectClient {

    Socket serverConn;
    ObjectInputStream inStream = null;
    ObjectOutputStream outStream = null;

    public ObjectClient(String host, int port) throws
        UnknownHostException, IOException {
        serverConn = new Socket(host, port);
        outStream = new ObjectOutputStream(serverConn.
            getOutputStream());
        inStream = new ObjectInputStream(serverConn.
            getInputStream());
    }
```

```
public Object sendMessage(int message_id, Object
    message) throws Exception {
    outStream.writeInt(message_id);
    outStream.writeObject(message);
    outStream.flush();
    Object response = inStream.readObject();
    return response;
}

public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    boolean finished = false;
    try {
        ObjectClient client = new ObjectClient("
            localhost", 3000);
        while (!finished) {
            System.out.println("\n\n1. Ktora godzina?
                ");
            System.out.println("2. Suma liczb w
                tablicy");
            System.out.println("3. Modyfikacja tekstu
                ");
            System.out.println("0. Koniec");
            char c = scan.nextLine().charAt(0);
            switch (c) {
                case '1':
                    Date d = (Date) client.
                        sendMessage(1, null);
                    System.out.println("Server: " + d
                        );
                    break;
                case '2':
                    int[] data = new int[10];
                    for (int i = 0; i < 10; i++) {
                        data[i] = i + 1;
                    }
                    Integer response = (Integer)
                        client.sendMessage(2, data);
                    System.out.println("Server: " +
                        response);
                    break;
                case '3':
                    System.out.print("Wpisz tekst: ")
                        ;
                    String str = scan.nextLine();
```

```
                        String r = (String) client.
                            sendMessage(3, str);
                        System.out.println("Server: " + r
                            );
                        break;
                    case '0':
                        System.out.println("Server says: "
                            + (String)client.sendMessage
                            (0, null));
                        client.serverConn.close();
                        finished = true;
                        break;
                }
            }
        } catch (Exception e) {
            System.out.println("Exception:" + e);
            e.printStackTrace();
        }

    }
}
```

4 Zadanie

Stwórz aplikację klient-serwer, która pozwoli na zachowanie kopii zapasowej plików klienta (z wybranego katalogu, który może być podany bezpośrednio w kodzie) po stronie serwera, oraz na przywrócenie kopii z serwera do klienta. Serwer powinien móc obsłużyć następujące żądania klienta:

- czy plik x znajduje się na serwerze?
- czy plik x znajdujący się serwerze jest taki sam jak u klienta (to można stwierdzić porównując sumy kontrolne plików)?
- odebranie pliku od klienta i zapisanie go w określonym miejscu na dysku
- przesłanie pliku klientowi (klient powinien zapisać ten plik w określonej lokalizacji)

Po napisaniu serwera, rozszerz go tak, żeby mogło z niego korzystać wielu klientów. Dodatkowo, dopisz do klienta funkcjonalność umożliwiającą wyświetlenie raportu wskazującego które pliki mają swoją kopię na serwerze wraz z informacją czy jest ona identyczna z tą u klienta. Zastanów się jak możnaby obsłużyć usuwanie plików.