Технически университет – София
Факултет по Компютърни системи и управление

ПИК 3 - Java

Семинарни упражнения

гл. ас. д-р Антония Ташева

Резултати от контролното

ТРАГИЧНИ СА!

(Файл)

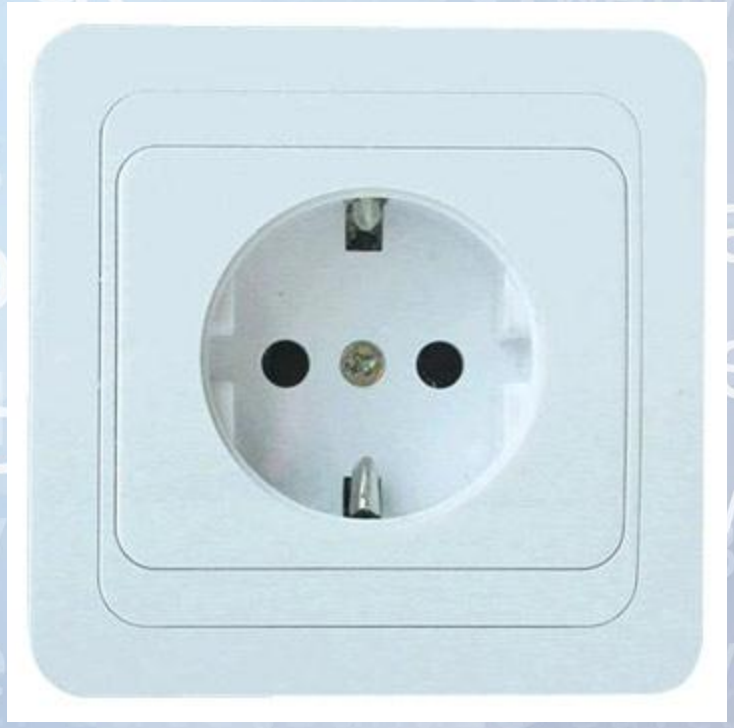
Домашни задания №4

– Само самостоятелни задачи – в края на часа

- 09.12 – няма да има час
- 10.12 (9:30 – 11:15 + 13:45 – 15:30) – предаване на Дом. 3
- 16.12 – лабораторно Сокети
- 06.01 – лабораторно Swing
- Предаване и защита на Домашно 4 - 14-та седмица (14.01.15)!

ТЕМА 7. СОКЕТИ

Какво е Socket?



Ще Ви се...

Какво е Socket?

- Сокетът представлява едната крайна точка в двустранна комуникация между две програми работещи в мрежа.
- Сокет класовете се използват за да представят връзката между клиентска и сървърна програми.
- java.net предоставя два класа
  - Socket – имплементира клиентската страна на връзката
  - ServerSocket – сървърната част от

Крайна точка

- Крайна точка (endpoint) е комбинация от IP адрес и номер на порт.
- Всяка TCP конекция може да бъде уникално идентифицирана от своите 2 крайни точки. По този начин може да съществуват множество връзки между хоста и сървъра.

Какво е Socket?

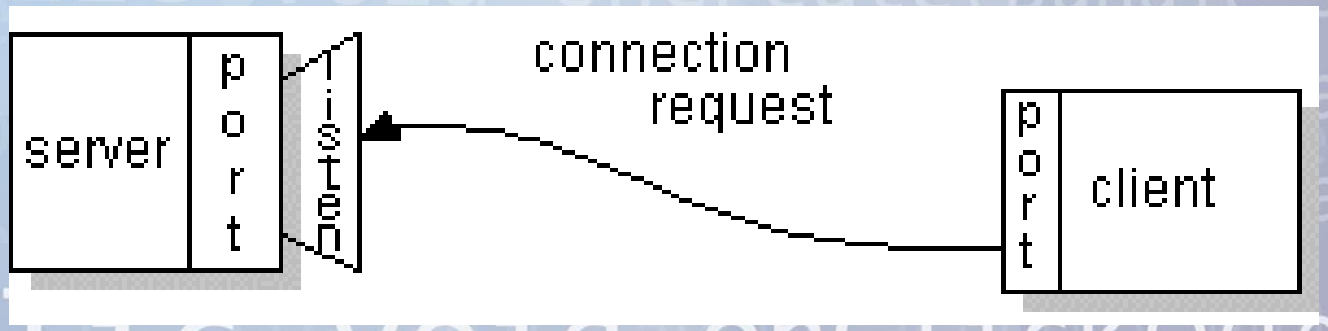
- Обикновено сървърната програма работи на определен компютър и е свързана с определен негов порт. Сървърът само изчаква, „слушайки“ този порт за клиентски заявки за връзка.

Какво е Socket?

- В клиентската страна:

Клиентът трябва да знае името на машината, на която работи сървъра и номера на порта, който той слуша.

За да се осъществи връзка са клиента подава заявка на този адрес. Необходимо е и той да се идентифицира, като това обикновено става автоматично от системата, като се задава локален порт.



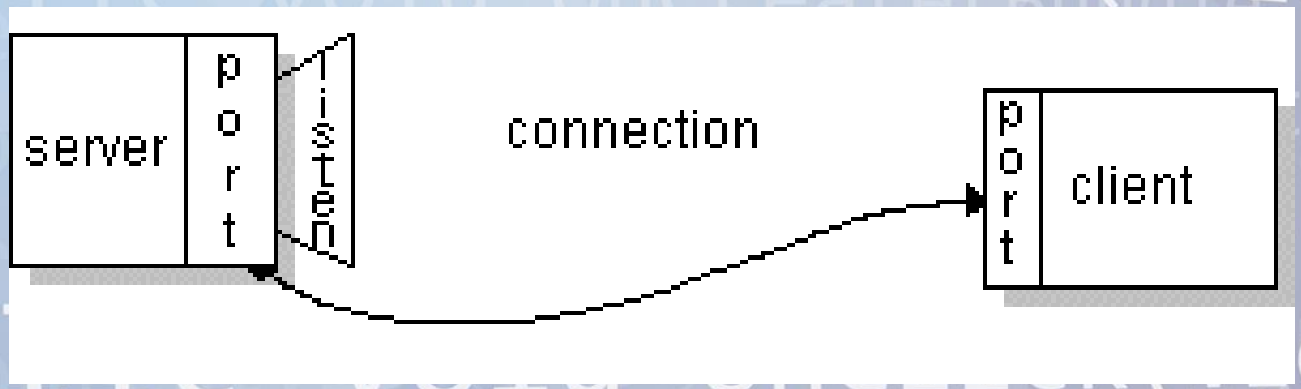
Какво е Socket?

- Ако всичко е наред, сървърът приема заявката и се осъществява връзка.
- При приемането, сървърът „връзва“ нов сокет с локалния порт и адреса на клиента.
- Необходим е нов сокет за да може сървърът да продължи да „слуша“ на оригиналния си порт за нови заявки за връзка, докато в същото време обслужва вече свързалите се клиенти.

Какво е Socket?

- От клиентската страна:

Ако се установи конекцията, сокетът се създава успешно и клиентът може да го използва за да си комуникира със сървъра.



- URLs and URLConnections provide a relatively high-level mechanism for accessing resources on the Internet. Sometimes your programs require lower-level network communication, for example, when you want to write a client-server application.
- In client-server applications, the server provides some service, such as processing database queries or sending out current stock prices. The client uses the service provided by the server, either displaying database query results to the user or making stock purchase recommendations to an investor. The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it.
- TCP provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.

Reading from and Writing to a Socket

- The example program implements a client, EchoClient, that connects to an echo server. The echo server receives data from its client and echoes it back. The example EchoServer implements an echo server.

(Alternatively, the client can connect to any host that supports the Echo Protocol.)

Пример

- Пълен пример и обяснение

- https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html

Reading from and Writing to a Socket

- The EchoClient example creates a socket, thereby getting a connection to the echo server. It reads input from the user on the standard input stream, and then forwards that text to the echo server by writing the text to the socket. The server echoes the input back through the socket to the client. The client program reads and displays the data passed back to it from the server.

```
String hostName = args[0];
int portNumber = Integer.parseInt(args[1]);

try (  // try-with-resources statement
   Socket echoSocket = new Socket(hostName, portNumber);
   PrintWriter out =
      new PrintWriter(echoSocket.getOutputStream(), true);
   BufferedReader in =
      new BufferedReader(
         new InputStreamReader(echoSocket.getInputStream()));
   BufferedReader stdIn =
      new BufferedReader(
         new InputStreamReader(System.in))
)
```

These lines establish the socket connection between the client and the server and open a PrintWriter and a BufferedReader on the socket.

The next interesting part of the program is the while loop. The loop reads a line at a time from the standard input stream and immediately sends it to the server by writing it to the PrintWriter connected to the socket:

```
String userInput;
while ((userInput = stdIn.readLine()) != null) {
    out.println(userInput);
    System.out.println("echo: " + in.readLine());
}
```

The last statement in the while loop reads a line of information from the BufferedReader connected to the socket. The readLine method waits until the server echoes the information back to EchoClient. When readline returns, EchoClient prints the information to the standard output.

- The echo server implements a simple protocol. The client sends text to the server, and the server echoes it back. When your client programs are talking to a more complicated server such as an HTTP server, your client program will also be more complicated. However, the basics are much the same as they are in this program:

  1. Open a socket.
  2. Open an input stream and output stream to the socket.
  3. Read from and write to the stream according to the server's protocol.
  4. Close the streams.
  5. Close the socket.

- Only step 3 differs from client to client, depending on the server. The other steps remain largely the same.

Writing the Server Side of a Socket

- **Server**: "Knock knock!"
  Client: "Who's there?"
  Server: "Dexter."
  Client: "Dexter who?"
  Server: "Dexter halls with boughs of holly."
  Client: "Groan."

The Knock Knock Server

- The server program creates the ServerSocket object in a try-with-resources statement:

```
int portNumber = Integer.parseInt(args[0]);

try (
   ServerSocket serverSocket = new ServerSocket(portNumber);
   Socket clientSocket = serverSocket.accept();
   PrintWriter out =
      new PrintWriter(clientSocket.getOutputStream(), true);
   BufferedReader in = new BufferedReader(
      new InputStreamReader(clientSocket.getInputStream()));
) {
```

The accept method

- The accept method waits until a client starts up and requests a connection on the host and port of this server. (Let's assume that you ran the server program KnockKnockServer on the computer named knockknockserver.example.com.) In this example, the server is running on the port number specified by the first command-line argument. When a connection is requested and successfully established, the accept method returns a new Socket object which is bound to the same local port and has its remote address and remote port set to that of the client. The server can communicate with the client over this new Socket and continue to listen for client connection requests on the original ServerSocket This particular version of the program doesn't listen for more client connection requests. However, a modified version of the program is provided in Supporting Multiple Clients.

Communicating with client

```
try (
   // ...
   PrintWriter out  = new PrintWriter(clientSocket.getOutputStream(), true);
   BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
) {
   String inputLine, outputLine;

   // Initiate conversation with client
   KnockKnockProtocol kkp = new KnockKnockProtocol();
   outputLine = kkp.processInput(null);
   out.println(outputLine);

   while ((inputLine = in.readLine()) != null) {
      outputLine = kkp.processInput(inputLine);
      out.println(outputLine);
      if (outputLine.equals("Bye."))
         break;
   }
```

- This code does the following:
  - Gets the socket's input and output stream and opens readers and writers on them.
  - Initiates communication with the client by writing to the socket (shown in bold).
  - Communicates with the client by reading from and writing to the socket (the while loop).
- Step 1 is already familiar. Step 2 is shown in bold and is worth a few comments. The bold statements in the code segment above initiate the conversation with the client. The code creates a KnockKnockProtocol object—the object that keeps track of the current joke, the current state within the joke, and so on.

Връзки

- **Java - Networking (Socket Programming)**

http://www.tutorialspoint.com/java/java_networking.htm

- **Socket Communications**

http://www.oracle.com/technetwork/java/socket-140484.html

ТЕМА 8. ГРАФИЧЕН ИНТЕРФЕЙС

Swing

- Swing е набор от пакети за изграждане на програми с графична среда на Java.
- Самият Swing е написан на езика Java, т.е. той е платформено независим. Част е от т.нар. „Java Foundation Classes“.

Библиотеки и класове

- import javax.swing.SwingUtilities;
- import javax.swing.JFrame;
- import javax.swing.JButton;
- import javax.swing.JPanel;

- …
- Събития …

Домашни задания №4

– Само самостоятелни задачи – в края на часа

- 09.12 – няма да има час
- 10.12 (9:30 – 11:15 + 13:45 – 15:30) – предаване на Дом. 3
- 16.12 – лабораторно Сокети
- 06.01 – лабораторно Swing
- Предаване и защита на Домашно 4 - 14-та седмица (14.01.15)!