

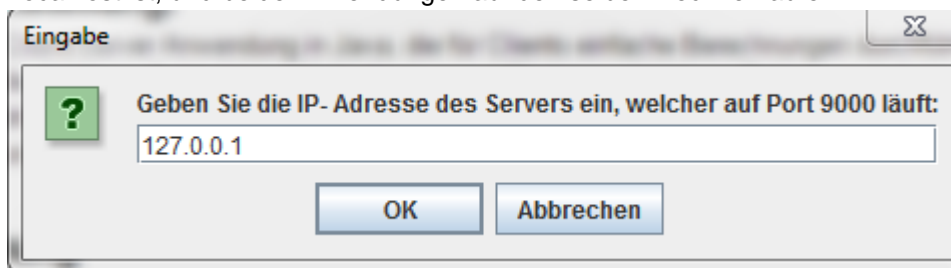
Protokoll Aufgabe 5:

Aufgabenstellung:

Realisiere eine Client/Server Anwendung in Java, die einen File-Transfer vom Server zum Client ermöglicht. Der Client kann irgendeinen Pfad zu einer Datei oder einem Ordner angeben der dann vom Server zum Client übertragen wird. Dabei sollen auch parallele Downloads möglich sein.

Handhabung

Das Programm besteht aus zwei ausführbaren Dateien, eine Serverdatei und eine Client Datei. Wichtig ist, dass der Server vor dem Client ausgeführt wird, sonst kommt es zu Problemen. Sollte man es geschafft haben, die Dateien in richtiger Reihenfolge auszuführen, muss man im Client als ersten Schritt IP-Adresse eingeben. Zum testen, wird die IP-Adresse 127.0.0.1 genutzt, da dies der Localhost ist, und beide Anwendungen auf den selben Rechner laufen.



Eingabe der IP-Adresse

Nachdem man dies erledigt hat, kommt ein neues Fenster, welches den Pfad der zu transferierenden Datei anfordert.

Der Ablauf

Nachdem der Server erstellt wurde, bietet er dem Client ein ServerSocket an, mithilfe dessen sich der Client zum Server verbinden kann. Der Server wird über Port 9000 erreichbar sein.

```
ServerSocket listener = new ServerSocket(9000);  
Socket socket = listener.accept();
```

Jetzt wartet der Server solange, bis sich ein Client verbindet. sollte sich ein Client verbinden wird dies durch listener.accept(); akzeptiert und zugelassen.

Im Client wird auch ein Socket erstellt, auch dies benutzt den Port 9000.

```
Socket s = new Socket(serverAddress, 9000);
```

Um etwas zu senden und zu empfangen, wurde jetzt eine abstrakte Klasse namens Message implementiert.

```

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

public abstract class Message implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 312853181604924426L;
    public void send(ObjectOutputStream writer) {
        try {
            writer.writeObject(this);
            writer.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static Message fromStream(ObjectInputStream reader) {
        try {
            return (Message) reader.readObject();
        } catch (ClassNotFoundException | IOException e) {
            e.printStackTrace();
            return null;
        }
    }
}

```

In der Klasse Message, werden zwei Methoden implementiert, welche für uns ein Objekt versenden und empfangen. Um ein Objekt zu versenden wird die Methode send zur Verfügung gestellt. Um Ein Objekt zu lesen bzw. zu empfangen wird uns die Methode from Stream zur Verfügung gestellt.

Um jetzt ein Objekt zu versenden sind folgende Codezeilen von nöten:

```

ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream());
ObjectInputStream in = new ObjectInputStream(s.getInputStream());
Authentifizierung auth = new Authentifizierung(getAuth());
auth.send(out);

```

(BEISPIEL)

Ein Objekt von Authentifizierung wird erstellt, dies muss von der Klasse Message erben, damit es diese nutzen kann. danach wird es mit auth.send(out); gesendet. Wichtig ist, dass es Die Authentifizierung im Server gibt, damit man dort mit den dort drin gespeicherten Daten arbeiten kann. Um ein Objekt zu lesen ist folgende Zeile von Nöten:

```
Authentifizierung a= (Authentifizierung) Message.fromStream(in);
```

(BEISPIEL)

Hier wird der Datentyp, welcher vorher verschickt wurde in ein schon existierendes Objekt Authentifizierung kopiert, damit man mit dieser Klasse arbeiten kann.

Um jetzt aber eine Datei zurückzusenden, kann man die klasse Message nicht mehr benutzen, da sie dafür nicht geeignet ist. Um etwas zurück zu senden wird folgender Code benötigt:

Server:

```
OutputStream out = s.getOutputStream();
InputStream fileToSend = new FileInputStream("ServerToSend.html");

byte[] buffer = new byte[1024];
while (fileToSend.available() > 0) {
    out.write(buffer, 0, fileToSend.read(buffer));
    System.out.println("...");
}
```

Anhand eines OutputStreams wird die Datei gesendet.

Client:

```
InputStream in = s.getInputStream();
FileOutputStream fileOut = new FileOutputStream("Download.html");

byte[] buffer = new byte[1024];
while (s.isConnected()) {
    int bytesRead = in.read(buffer);
    if (bytesRead == -1) break;
    fileOut.write(buffer, 0, bytesRead);
}
```

Anhand InputStream wird die Seite gelesen. Mit Hilfe von FileOutputStream wird die Seite in eine Datei geschrieben.

Um mehrere Downloads gleichzeitig zu garantieren werden Threads benötigt.

In der Methode Thread wird ein Threadpool erstellt. Durch execute wird so ein Thread des Pools aktiviert.

```
public void start() {

    try {
        ServerSocket listener = new ServerSocket(9000);
        ExecutorService executor = Executors.newCachedThreadPool();
        boolean weiter = true;

        Socket s = listener.accept();
        executor.execute(new SThread(s, this));
        listener.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

    System.out.println("Server beendet");
}
```

Durch die Methode run() kann ein Thread implementiert werden. Bei jedem neuem Datentransfer wird ein neuer Thread aus dem Pool genommen, welcher dann die run()- Methode ausführt.. Die Klasse, welche diese Methode enthält muss von Thread erben.

```

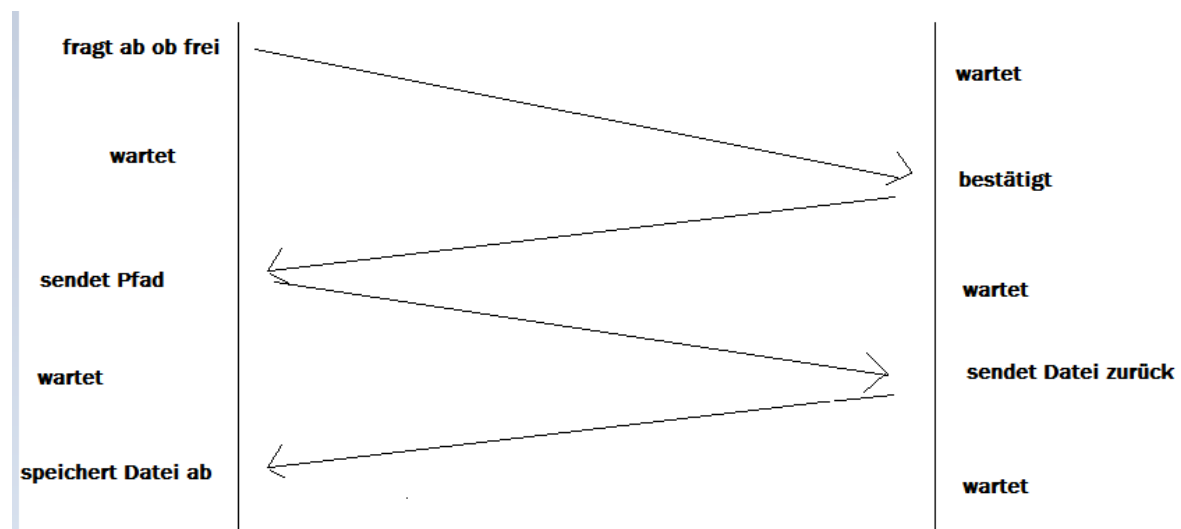
public void run(){
    try {
        ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
        try {
            ToSendGet buffpath = (ToSendGet)Message.fromStream(in);
            Path path = (Paths.get(buffpath.getPfad()));

            OutputStream out = socket.getOutputStream();
            InputStream toSend = new FileInputStream(path.toFile());
            byte[] buffer = new byte[1024];
            while (toSend.available() > 0) {
                out.write(buffer, 0, toSend.read(buffer));
                System.out.println("Gesendet");
            }
            toSend.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    } catch (Exception e) {
    } finally {
        try{
            System.out.println("Gesendet");
            socket.close();

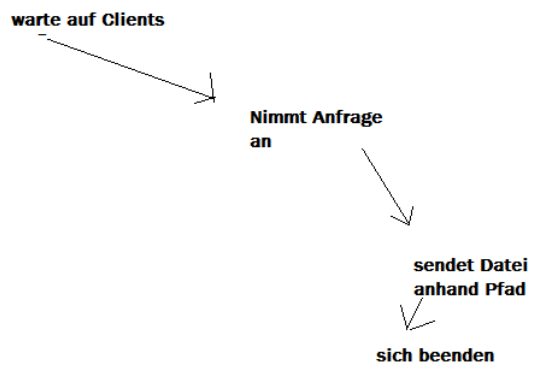
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Grafiken:



Server:



Client:

