



Implementierung eines Webclients



Klasse WebClient – Grundgerüst

- Öffne das BlueJ-Projekt „Webclient Vorlage“.
- Deklariere die drei Attribute. Scanner, PrintWriter und SSLSocket sind von Java vorgegebene Klassen, die man über die Imports aus der Java-Bibliothek in das eigene Programm einbindet.
- Der Konstruktor bleibt ausnahmsweise leer, denn die Attribute werden erst bei der Verbindungserstellung initialisiert.

WebClient
in : Scanner out : PrintWriter socket : SSLSocket
+ WebClient() + holeWebseite() : void + stelleVerbindungHer() : void + verbindungTrennen() : void - sendeNachricht(msg : String) : void

Klasse Client – stelleVerbindungHer()

- Setze sinnvolle Werte für Servername und Port ein. Code für die Verbindungsherstellung:

```
SSLConnectionFactory factory = (SSLConnectionFactory)
SSLConnectionFactory.getDefault();
```

```
socket = (SSLSocket) factory.createSocket(<Servername>, <port>);
```

```
out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream())));
```

```
in = new Scanner(new BufferedReader(new
InputStreamReader(socket.getInputStream())));
```

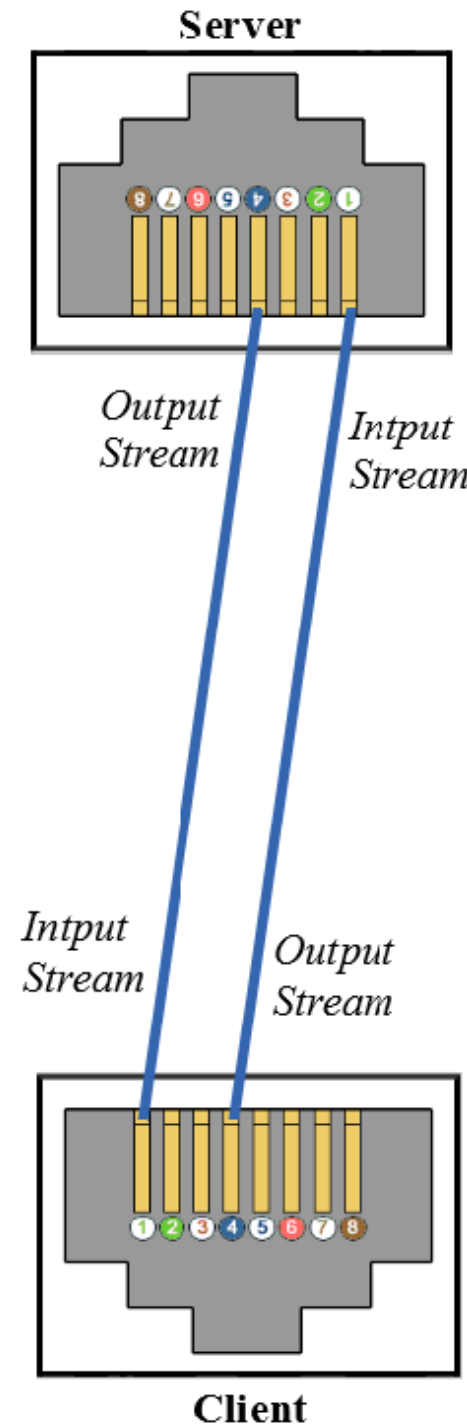
Erklärung zur Initialisierung der Verbindung

- Die SSLSocketFactory ist eine Fabrik zum Erstellen verschiedener SSLSocket-Objekte mit unterschiedlicher Konfiguration. Man sagt der Fabrik, was der SSLSocket können soll und die Fabrik liefert dann einen passend eingestelltes SSLSocket-Objekt zurück.
- SSLSocketFabrik erzeugen (hier Standard/Default-Fabrik):
`SSLSocketFactory factory = (SSLSocketFactory)
SSLSocketFactory.getDefault();`
- Passend konfigurierten Socket erhalten (hier nur mit Server- und Portangabe; aber man kann noch viel mehr konfigurieren):
`socket = (SSLSocket) factory.createSocket(<Servername>, <port>);`

Erklärung zur Initialisierung der Verbindung

- Jeder Socket besitzt einen `InputStream` und einen `OutputStream`. Wenn sich zwei Sockets verbunden haben, dann werden die Daten/Nachrichten über den Output des einen Sockets versendet und landet beim anderen Socket im Input und auch umgekehrt. (siehe Bild)
- Der `InputStream` und `OutputStream` arbeiten auf **Bitebene**. Das heißt, es werden beim `InputStream` **Bytes** gelesen/empfangen und beim `OutputStream` **Bytes** geschrieben/versendet.

```
out = new PrintWriter(new BufferedWriter(new  
OutputStreamWriter(socket.getOutputStream())));  
in = new Scanner(new BufferedReader(new  
InputStreamReader(socket.getInputStream())));
```



Erklärung zur Initialisierung des In-/Out-Kanals

- Die Klassen `InputStreamReader` und `OutputStreamWriter` ermöglichen es, dass man die Daten **zeichenweise** (Buchstaben, Zahlen und Sonderzeichen nach einem spezifizierten Zeichensatz) lesen/schreiben kann.

```
out = new PrintWriter(new BufferedWriter(new  
OutputStreamWriter(socket.getOutputStream())));
```

```
in = new Scanner(new BufferedReader(new  
InputStreamReader(socket.getInputStream())));
```

Erklärung zur Initialisierung des In-/Out-Kanals

- Die Klasse `BufferedReader` kann einen zeichenbasierten Stream lesen und puffert/speichert die gelesenen Zeichen zwischen, sodass man anschließend z.B. **zeilenweise** die Daten/Texte lesen kann.
- Die Klasse `BufferedWriter` puffert Zeichen(-folgen) (also Text), um effizientes Schreiben (z. B. **zeilenweise**) in einen zeichenbasierten Outputstream zu ermöglichen.

```
out = new PrintWriter(new BufferedWriter(new  
OutputStreamWriter(socket.getOutputStream())));
```

```
in = new Scanner(new BufferedReader(new  
InputStreamReader(socket.getInputStream())));
```

Erklärung zur Initialisierung des In-/Out-Kanals

- Die Klasse `PrintWriter` bietet die `print(...)`- bzw. `println(...)`-Methoden für primitive **Datentypen** und **Strings** an.
- Der `Scanner` kann die eingelesenen Daten/Texte passend „zerteilen“ und für die Weiterverarbeitung in den ursprünglichen **Datentyp** zurückwandeln.
(Stichwort: Parser/Parsing)

```
out = new PrintWriter(new BufferedWriter(new  
OutputStreamWriter(socket.getOutputStream())));
```

```
in = new Scanner(new BufferedReader(new  
InputStreamReader(socket.getInputStream())));
```

Klasse WebClient – Methode `sendeNachricht(String msg)`

- Gib die Nachricht aus, sodass man in der Konsole auch lesen kann, was der Client sagt. ->
`System.out.println();`
- Schreibe über die Methode `println()` des Printwriters ebenfalls in den out-Kanal.
- Schicke über die Methode `flush()` des out-Kanals los.

WebClient
in : Scanner out : PrintWriter socket : SSLSocket
+ WebClient() + holeWebseite() : void + stelleVerbindungHer() : void + verbindungTrennen() : void - sendeNachricht(msg : String) : void

Klasse WebClient – Methode trenneVerbindung()

- Schließe zuerst die beiden Kanäle (in und out) mittel der Methode **close()**
- Schließe zum Schluss den Socket ebenfalls mit der Methode **close()**

WebClient
in : Scanner out : PrintWriter socket : SSLSocket
+ WebClient() + holeWebseite() : void + stelleVerbindungHer() : void + verbindungTrennen() : void - sendeNachricht(msg : String) : void

Klasse WebClient – Methode `holeWebseite()`

- Nutze die selbst geschriebene Methode `sendeNachricht()`, um analog wie bei openssl die HTTP-GET-Anfrage zu gestalten. (siehe Skript)
- Nimm die Antwort des Servers entgegen, indem:
 - Solange im in-Kanal eine weitere Zeile zu finden ist (Methode `hasNextLine()`), dann soll diese Zeile (Methode `nextLine()`) mittels `System.out.println()` ausgegeben werden.

WebClient
in : Scanner out : PrintWriter socket : SSLSocket
+ WebClient() + holeWebseite() : void + stelleVerbindungHer() : void + verbindungTrennen() : void - sendeNachricht(msg : String) : void

Klasse Webclient – Testen

- Teste deinen Webclient, indem du einen Webclient erstellst, eine Verbindung aufbaust, die Webseite holst und dann die Verbindung schließt.