Dokumentation der Klasse Account:

Konstruktor	`Account(String Name, String Passwort, String Email)`
	Ein Objekt der Klasse Account wird mit den vorher deklarierten Attributen: `Name` (String), `Passwort`, (String), `Email` (String), `Basket` (List, welche Artikel verwaltet), `Wishlist` (List, welche Artikel verwaltet), `Notifications` (List, welche Strings verwaltet) initialisiert, hierbei sind Name, Email und Passwort Parameter, der Nutzer muss also seinen Name, eine E-Mail Adresse und ein Passwort angeben.
Anfrage	`String getName()`
	Der Name eines Accounts wird als String ausgegeben.
Auftrag	`void setName(String NameNew)`
	Der Name eines Accounts wird mit dem als Parameter angegebenen Namen ersetzt.
Anfrage	`String getPasswort()`
	Das Passwort eines Accounts wird als String ausgegeben.
Auftrag	`void setPasswort(String PasswortNew)`
	Das Passwort eines Accounts wird mit dem als Parameter angegebenen Passwort ersetzt.
Anfrage	`List<Artikel> getBasket()`
	Der Basket eines Accounts wird als Liste, welche Artikel enthält ausgegeben.
Auftrag	`void setBasket(List<Artikel> BasketNew)`
	Der Basket eines Accounts wird mit dem als Parameter angegebenen Basket ersetzt.
Anfrage	`String getEmail()`
	Die Email eines Accounts wird als String ausgegeben.
Auftrag	`void setEmail(String EmailNew)`
	Die Email eines Accounts wird mit der als Parameter angegebenen Email ersetzt.
Auftrag	`void clearbasket()`
	Der Basket eines Accounts wird mit einem neu erzeugten Basket ersetzt, wodurch für den Nutzer der Basket entleert scheint.
Anfrage	`List<Artikel> getWishlist()`
	Die Wishlist eines Accounts wird als Liste, welche Artikel enthält ausgegeben.
Auftrag	`void clearwishlist()`

Die Wishlist eines Accounts wird mit einer neu erzeugten Wishlist ersetzt, wodurch für den Nutzer die Wishlist entleert scheint.

Auftrag `void addtobasket(Artikel partikel)`

Wenn der übergeben Artikel in Form eines Parameters nicht das Nullobjekt ist, wird dieser an die Liste Basket angehängt.

Auftrag `void addtowishlist(Artikel partikel)`

Wenn der übergeben Artikel in Form eines Parameters nicht das Nullobjekt ist, wird dieser an die Liste `Wishlist` angehängt.

Auftrag `void removefrombasket(int partikelnummer)`

Der Basket wird solange durchlaufen, bis entweder ein Artikel welches die als Parameter übergebene `Artikelnummer` trägt, entfernt wurde oder das Ende der Liste erreicht wurde.

Auftrag `void removefromwishlist(int partikelnummer`)

Die `Wishlist` wird solange durchlaufen, bis entweder ein Artikel welches die als Parameter übergebene Artikelnummer trägt, entfernt wurde oder das Ende der Liste erreicht wurde.

Auftrag `void deletenotifications ()`

Die Notifications eines Accounts werden mit einer neu erzeugten Liste von Notifications ersetzt, wodurch für den Nutzer die Notifications gelöscht scheinen.

Auftrag `void notify (String pstring)`

Wenn der übergeben String in Form eines Parameters nicht das Nullobjekt ist, wird dieser an die Liste Notifications angehängt.

Anfrage `List<String> getNotifications()`

Die Notifications eines Accounts werden als Liste, welche Strings enthält ausgegeben.

Dokumentation der Klasse Artikel:

Konstruktor `Artikel(String Name, int Artikelnummer, int Preis, String Beschreibung, String Hersteller)`

Ein Objekt der Klasse Artikel wird mit den vorher deklarierten Attributen: `Name` (String), `Artikelnummer` (int), `Preis` (int), `Beschreibung` (String), `Hersteller` (String) und `Verkaufszahl` (int) initialisiert, hierbei sind Name, Artikelnummer, Preis, Beschreibung und Hersteller Parameter.

Anfrage `String getName()`

Der Name eines Artikels wird als String ausgegeben.

Anfrage	`int getPreis()`
	Der Preis eines Artikels wird als integer ausgegeben.
Anfrage	`int getArtikelnummer()`
	Die Artiklenummer eines Artikels wird als integer ausgegeben.
Anfrage	`String getBeschreibung()`
	Die Beschreibung eines Artikels wird als String ausgegeben.
Anfrage	`String getHersteller()`
	Der Hersteller eines Artikels wird als String ausgegeben.
Auftrag	`void verkauft()`
	Die Verkaufszahl eines Artikels wird um eins erhöht.
Anfrage	`int getverkaufszahl()`
	Die Verkaufszahl eines Artikels wird als integer ausgegeben.

Dokumentation der Klasse Bestellung:

Konstruktor	`Bestellung(Artikel artikel, int Bestellungsnummer, Account Käufer)`
	Ein Objekt der Klasse Bestellung wird mit den vorher deklarierten Attributen: `Artikel` (Artikel), `Bestellungsnummer` (int), und `Käufer` (Account) initialisiert, hierbei sind Artikel, Bestellungsnummer und Käufer Parameter.
Anfrage	`Artikel getArtikel()`
	Der Artikel einer Bestellung wird als Artikel ausgegeben.
Anfrage	`int getBestellungsnummer()`
	Die Bestellungsnummer einer Bestellung wird als integer ausgegeben.
Anfrage	`Account getKäufer()`
	Der Käufer einer Bestellung wird als Account ausgegeben.
Anfrage	`String getStatus()`
	Der Status einer Bestellung wird als String ausgegeben.
Auftrag	`void setStatus(String StatusNew)`
	Der Status einer Bestellung wird mit dem als Parameter angegebenen Status ersetzt.

Dokumentation der Klasse Bestellungen:

Konstruktor	`Bestellungen ()`

Ein Objekt der Klasse Bestellungen wird mit den vorher deklarierten Attributen: `Bestellungen` (List, welche Objekte der Klasse Bestellung verwaltet) und Bestellungsnummercounter (int) initialisiert.

Anfrage `List<Bestellung> getBestellungen()`

Die Bestellungen eines Accounts werden als Liste, welche Objekte der Klasse Bestellung enthält ausgegeben.

Auftrag `void setBestellungen(List<Bestellung> bestellungenNew)`

Die Bestellungen eines Accounts werden mit der als Parameter angegebenen Liste, welche Objekte der Klasse Bestellung enthält, ersetzt.

Anfrage `int getBesttellungsnummecounter()`

Der Bestellungsnummecounter eines Objektes der Klasse Bestellungen wird als integer ausgegeben.

Auftrag `void bestellen(Artikel partikel, Account paccount)`

Es wird eine neue Bestellung temp erstellt mit den Attributen, welche als Parameter angegeben sind. Der Dienst verkauft wird mit dem als Parameter angegebenen Artikel durchgeführt, der Bestellungsnummercounter wird um eins erhöht und temp wird an die Liste der Bestellungen angehängt.

Anfrage `List<Bestellung> allordersfromaccount(Account pacc)`

Es wird die Liste aller eingegangenen Bestellungen, von vorne bis hinten durchlaufen. Hierbei wird jede Bestellung, dessen Account mit dem Account im Parameter übereinstimmt in temp (eine lokale Liste mit Objekten der Klasse Bestellung) angehängt. Sobald die gesamte Liste durchlaufen wurde, wird temp ausgegeben.

Dokumentation der Klasse Lager:

Konstruktor `public Lager()`

Ein Objekt der Klasse Lager wird mit den vorher deklarierten Attributen: `myitems` (Liste, welche Artikel verwaltet) und `Artikelnummercounter` (int) initialisiert.

Anfrage `List<Artikel> getMyitems()`

Die `myitems` eines Accounts werden als Liste, welche Artikel enthält ausgegeben.

Auftrag `void setMyitems(List<Artikel> myitemsNew)`

Die `myitems` eines Accounts werden mit der als Parameter angegebenen Liste, welche Artikel enthält, ersetzt.

Anfrage `List<Artikel> searchitem(String pname)`

Es wird die Liste aller im Lager vorhandenen Artikel von vorne bis hinten durchlaufen. Hierbei wird jeder Artikel, dessen Name mit dem Namen im Parameter übereinstimmt in temp (eine lokale Liste mit Artikeln) angehängt. Sobald die gesamte Liste durchlaufen wurde, wird temp ausgegeben.

Anfrage `Artikel searchitemtbynumber(int number)`

Es wird die Liste aller im Lager vorhandenen Artikel von vorne bis hinten durchlaufen. Hierbei wird der Artikel, dessen Artikelnummer mit der Nummer im Parameter übereinstimmt ausgegeben.

Auftrag `void additem(String pname, int ppreis, String Beschreibung, String Hersteller)`

Es wird ein Artikel temp deklariert und initialisiert, welcher die Attribute, welche als Parameter übergeben sind trägt. Dieser Artikel wird an die Liste des Lagers angehangen, wobei der artikelnummercounter um eins erhöht wird.

Auftrag `void sort()`

Die Liste aller im Lager vorhandenen Artikel wird mit dem Selectionsort nach verkaufszahl sortiert.

Anfrage `List<Artikel> nmostpopular(int n)`

Zunächst wird der Dienst void sort() genutzt um die Liste aller im Lager vorhandenen Artikel nach verkaufszahl, und somit Beliebtheit sortiert. Dann wird die Liste solange durchlaufen und jedes aufgerufene Objekt in eine lokale Liste temp verschoben, bis die Anzahl der Objekte in temp den als Parameter angegebenen integer erreicht hat. Zuletzt wird temp ausgegeben.

Dokumentation der Klasse User:

Konstruktor `User(String Ip, int Port)`

Ein Objekt der Klasse User wird mit den vorher deklarierten Attributen: `Ip` (String), `Port` (int), und `myaccount` (Account) initialisiert, hierbei sind Ip, und Port Parameter.

Anfrage `String getIp()`

Die Ip eines Users wird als String ausgegeben.

Auftrag `void setIp(String IpNew)`

Die Ip eines Users wird mit der als Parameter angegebenen Ip ersetzt.

Anfrage `int getPort()`

Der Port eines Users wird als integer ausgegeben.

Auftrag `void setPort(int PortNew)`

Der Port eines Users wird mit dem als Parameter angegebenen Port ersetzt.

Anfrage `Account getMyaccount()`

Der myaccount eines Users wird als Account ausgegeben.

Auftrag `void setMyaccount(Account myaccountNew)`

Der Account eines Users wird mit dem als Parameter angegebenen Account ersetzt.

Dokumentation der Klasse Userbank:

Konstruktor `Userbank()`

Ein Objekt der Klasse Userbank wird mit dem vorher deklarierten Attribut: users (Liste, welche Accounts enthält) initialisiert.

Anfrage `List<Account> getUsers()`

Die users einer `Userbank` werden als Liste, welche Accounts enthält ausgegeben.

Auftrag `void adduser(String name, String passwort, String email)`

Es wird ein Account temp deklariert und initialisiert, welcher die Attribute, welche als Parameter übergeben sind trägt. Wenn der erzeugte Account nicht das Nullobjekt ist, wird dieser an die Liste users angehängt.

Anfrage `Account searchbyname(String pname)`

Es wird die Liste aller in users vorhandenen Accounts von vorne bis hinten durchlaufen. Hierbei wird der Account, dessen Name mit dem Namen im Parameter übereinstimmt ausgegeben.

Dokumentation der Klasse Onlineshop Server

Konstruktor `Onlineshop_Server()`

Ein Objekt vom Typ `Onlineshop_Server()` wird erstellt, das ein vorher festgelegtes Parameter besitzt, welches in diesem Fall den Port des Servers darstellt. Zudem besitzt die Klasse die Attribute Userbank, Lager, Bestellungen eine Liste, die User verwaltet und eine Liste, die Befehle in Form von Strings verwaltet.

Auftrag `void processNewConnection(String pClientIP, intpClientPort)`

Diese Methode wird aufgerufen, wenn sich ein Client mit IP-Adresse `pClientIP` und Portnummer `pClientPort` mit dem Server verbunden hat. Sie informiert den Kunden, dass er mit dem Onlineshop verbunden ist und fordert ihn auf, sich anzumelden.

Auftrag `void processMessage(String pClientIP, int pClientPort, String pMessage)`

Diese Methode wird aufgerufen, wenn der Server die Nachricht `pMessage` von dem durch `pClientIP` und `pClientPort` spezifizierten Client empfangen hat. Dabei werden unterschiedliche Fälle abgedeckt.

`case "?LOGIN"`

Hierbei wird verglichen, ob das eingegebene Passwort `(input[2])` zu dem angegebenen Accountnamen `(input[1])` passt. Wenn dies der Fall ist, wird der Client benachrichtigt, dass er erfolgreich eingeloggt ist. Wenn nicht, dann wird er benachrichtigt, dass das Einloggen fehlgeschlagen ist.

`case "?BASKET"`

Hierbei wird der Dienst `Artikelformat` verwendet, um den `basket` des Accounts auszugegeben. (siehe `String Artikelformat`)

`case "?SEARCH"`

Hierbei wird in der List `myitems` der Klasse `Lager` nach dem gesuchten Artikel (input[1]) gesucht. Wenn dieser gefunden wurde, wird er asn den Client ausgegeben.

`case "?ADDTOBASKET"`

Hierbei wird der ausgewählte Artikel dem `basket` des Accounts hinzugefügt. Dazu wird der Dienst `addtobasket` der Klasse `Account` verwendet. Der Client wird sowohl informiert, wenn der Artikel erfolgreich hinzugefügt wurde, als auch wenn das Hinzufügen des Artikels fehlgeschlagen ist.

case "?REMOVEFROMBASKET"

Hierbei wird ein Artikel aus dem `basket` entfernt. Dazu wird der Dienst `removefrombasket` der Klasse `Account` verwendet. Der Client wird sowohl informiert, wenn der Artikel erfolgreich entfernt wurde, als auch wenn das Entfernen des Artikels fehlgeschlagen ist.

case "?NEWACCOUNT"

Hierbei wird ein neuer Account erstellt. Dazu wird der Dienst `adduser` der Klasse `Userbank` verwendet. Der Client wird sowohl informiert, wenn der neue Account erfolgreich erstellt wurde, als auch wenn das Erstellen des Accounts fehlgeschlagen ist.

case "?LOGOUT&BUY"

Hierbei werden alle Artikel, die sich im `basket` befinden mit Hilfe des Dienstes `bestellen` der Klasse `Bestellungen` bestellt. Wenn dies erfolgreich geschehen ist, wird der Client benachrichtigt und die Verbindung wird getrennt.

case "?CLEARBASKET"

Hierbei wird durch den Dienst `clearbasket` der Klasse `Account` ein neuer `basket` erstellt. Der Client wird anschließend darüber informiert.

case "?HISTORY"

Hierbei werden alle Bestellungen, die über den Account vollzogen worden sind, ausgegeben. Dazu wird der Dienst `allordersfromaccount` der Klasse `Bestellungen` verwendet.

case "?NOTIFICATIONS"

Hierbei werden alle `Notifications` des Accounts ausgegeben. Dazu wird der Dienst `Notificationformat` der Klasse `Onlineshop Server` verwendet.

case "?DELETENOTIFICATIONS"

Hierbei werden alle Notifications gelöscht. Dazu wird der Dienst `deletenotifications` der Klasse `Account` verwendet. Wenn der Dienst erfolgreich ausgeführt wurde, wird der Client darüber informiert. Wenn nicht, wird er ebenfalls benachrichtigt.

case "?WISHLIST"

Hierbei wird die Liste `Wishlist` des Accounts ausgegeben. Wenn dabei ein Fehler auftritt, wird der Client darüber informiert.

case "?ADDTOWISHLIST"

Hierbei können Artikel über ihre Artikelnummer zu der `Wishlist` hinzugefügt werden. Dabei wird auf den Dienst `addtowishlist` der Klasse `Account` zurückgegriffen. Der Client wird sowohl informiert, wenn der Artikel erfolgreich hinzugefügt wurde, als auch wenn das Hinzufügen des Artikels fehlgeschlagen ist.

case "?REMOVEFROMWISHLIST"

Hierbei können Artikel aus der `Wishlist` entfernt werden. Dabei wird auf den Dienst `removefromwishlist` der Klasse `Account` zurückgegriffen. Der Client wird sowohl informiert, wenn der Artikel erfolgreich entfernt wurde, als auch wenn das Entfernen des Artikels fehlgeschlagen ist.

case "?CLEARWISHLIST"

Hierbei werden alle Artikel aus der `Wishlist` entfernt. Dabei wird auf den Dienst `clearwishlist` der Klasse `Account` zurückgegriffen.

case "?RECOMMEND"

Hierbei wird eine Liste der beliebtesten Artikel ausgegeben. Dabei wird auf den Dienst `nmostpopular` der Klasse `Lager` zurückgegriffen.

case "?ARTIKEL"

Hierbei wird eine Liste ausgegeben, die aus Artikeln besteht, die den eingegebenen Namen besitzen.

case "?ALLARTIKEL"

Hierbei werden alle Artikel, die im Onlineshop vorhanden sind, ausgegeben.

case "?COMMANDS"

Hierbei werden alle Befehle, die im Onlineshop zur Verfügung stehen, ausgegeben. Dabei wird auf den Dienst `getallcommands` der Klasse `Onlineshop Server` zurückgegriffen.

case "?QUIT"

Hierbei werden alle Objekte, die im Basket gespeichert sind, entfernt und die Verbindung zum Client wird getrennt.

Auftrag `void processClosingConnection(String pClientIP, int pClientPort)`

Diese Methode wird aufgerufen, wenn der Server die Verbindung zu dem durch `pClientIP` und `pClientPort` spezifizierten Client trennt. Hierbei durchläuft der Dienst die `List onlineusers` des Onlineshop Servers und vergleicht die `Ip` und den `Port` jeden Users mit den, im Methodenkopf übergebenen Parameter `String pClientIP` und `int pClientPort`. Wenn der richtige User gefunden wurde wird er aus der Liste entfernt und benachrichtigt, dass die Verbindung getrennt worden ist.

Anfrage `User getuserbyIpandPort(String ip, int port)`

Diese Anfrage wird verwendet, um einen User über die Parameter `String ip` und `int port` ausfindig zu machen. Hierbei durchläuft der Dienst die `List onlineusers` des Onlineshop Servers und vergleicht die `Ip` und den `Port` jeden Users mit den, im Methodenkopf übergebenen Parameter `String ip` und `int port`. Wenn der passenden User gefunden ist, wird dieser ausgegeben. Wenn der User in der Liste nicht vorhanden ist, wird `null` ausgegeben.

Auftrag `void removeuserswithaccount(Account pacc)`

Diese Methode wird aufgerufen, wenn ein User gelöscht werden soll. Hierbei durchläuft der Dienst die `List onlineusers` des Onlineshop Servers. Dabei wird bei jedem User verglichen ob es sich um den `Account` handelt, der im Methodenkopf mit dem Parameter `pacc` übergeben wird. Wenn dies der Fall ist, wird der User gelöscht. Wenn nicht, wird das Durchlaufen fortgeführt.

Anfrage `String Artikelformat (List<Artikel> plis)`

Diese Anfrage liefert den Warenkorb `basket`, der aus der `plis` mit Objekten des Datentyps `Artikel` besteht Hierbei werden mit Hilfe der Getter der Klasse `Artikel` Artikelnummer, Name, Beschreibung, Preis und Hersteller des Artikels ausgegeben.

Anfrage `String Bestellformat (List<Bestellung> plis)`

Diese Anfrage liefert den Warenkorb `basket`, der aus der `plis` mit Objekten des Datentyps `Bestellung` besteht. Hierbei werden mit Hilfe der Getter der Klasse `Artikel` und der Klasse `Bestellung` Artikelnummer, Name,

Beschreibung, Preis, Hersteller und Bestellungsnummer des bestellten Artikels ausgegeben.

Anfrage `String Notificationformat (List<String> plis)`

Diese Anfrage liefert eine Liste mit `notifications`, die jeder Account des Onlineshops besitzt (siehe Klasse Account). Diese sind in der `List plis` gespeichert, welche Objekte des Datentyps `String` verwaltet.

Anfrage `public String getallcommands()`

Dieser Dienst gibt alle Befehle aus, die in der Liste `commands` der Klasse `Onlineshop Server` gespeichert sind. Dazu wird diese durchlaufen.

Anfrage `public String allaccounts()`

Dieser Dienst gibt alle User aus, die in der Liste `userbank` gespeichert sind. Dabei wird die Liste durchlaufen und Name, Passwort und Email der User werden ausgegeben.

Anfrage `public String allarticles()`

Dieser Dienst gibt alle Artikel aus, die in der Liste `myitems` der Klasse `Lager` gespeichert sind. Ausgegeben werden hierbei Artikelnummer, Name, Beschreibung, Preis und Hersteller des Artikels.

Verknüpfung der GUI

Die GUI wird mit dem Server verbunden, indem in der GUI ein Client implementiert wird, der mit dem Server kommunizieren kann. In der Klasse des Clients werden nun Dienste wie Getter implementiert, die Informationen vom Server erfragen können. Diese Dienste können dann von der GUI aufgerufen werden, wodurch die Informationen des Servers an die GUI gelangen können (GUIClient.getXY()).

Netzwerke - Der Ablauf einer Session zwischen Server und Client:

Verbindungsaufbau seitens des Clients:

Dies geschieht in form des Aufrufs des Konstruktors, des Clients. Dieser versucht dann eine Verbindung zu dem durch die Parameter spezifizierten Server aufzubauen. Oder auch im Code:

```
16    public Onlineshop_Client(){
17      super("192.168.178.24",80);
18      }
```

Aufruf der Methode processNewConnection auf der Serverseite:
Verbindet sich ein neuer Client mit dem Server wird in diesem die Methode processNewConnection ausgeführt. In dieser hat der Server z.B. die Chance sich diesen neuen Client zu merken um ihn im späteren Verlauf der Session wiederzuerkennen. (Hierbei werden der Methode die Ip-Adresse, wie auch die Portnummer des Clients übergeben) Oder auch im Code:

```
26    public void processNewConnection(String pClientIP, int pClientPort){
27      User temp = new User(pClientIP,pClientPort);
28      onlineusers.append(temp);
29      this.send(pClientIP,pClientPort,"Hallo Kunde. Sie sind jetzt mit dem Onlineshop verbunden. Bitte melden sie sich nun an oder erstellen
30    }
```

!Wichtig! Für die gesamte Session werden nun Strings zwischen dem Client und dem Server wechselseitig geschickt. Dies ist besonders wichtig, damit die Integrität der Kommunikation gewährleistet werden kann. Und damit beide über den Status des anderen informiert sind. (Die Rückantwort dient hier als Bestätigung)

Kommunikation zwischen Server und Client (Aufforderungen/Anfragen):

Um von dem Server/Client die gewünschte Reaktion zu erhalten muss der Client/Server jeweils einen nach dem Protokoll (Regelbuch/Sprache in der Server und Client kommunizieren (mit strengen Regeln)) definierten String schicken, der zusätzlich die benötigten Parameter/Informationen enthält. Oder auch im Code:

```
56          break;
57        case "?BASKET" :
58          {
59
60            if (temp.getMyaccount()!=null) {
61              String back = "!BASKET"+this.Artikelformat(temp.getMyaccount().getBasket());
62              this.send(pClientIP,pClientPort,back);
63            } else {
64              this.send(pClientIP,pClientPort,"FAILURE");
65            } // end of if-else
66          }
67          break;
```

Hier ist zum Beispiel ein Befehl implementiert, der den Warenkorb des Clients an diesen im spezifizierten Format zurückschickt.

Beendigung der Session zwischen Client und Server:

Entweder wird die Verbindung auf Wunsch des Clients beendet, dies geschieht in dem dieser ein Keyword and den Server schickt, welcher dann durch eine letzte Nachricht die Abmeldung bestätigt und anschließend die Verbindung zu dem Client trennt. Wichtig für diesen Vorgang ist die Methode closeConnection(String pClientip, int pClientport). Diese ist zu Nutzen, um die Verbindung zu einem spezifischen Client zu beenden. Intern wird auf Serverseite auch noch die Methode processClosingConnection(String pClientip, int pClientport), die sich darum kümmert, dass der Client über den Verbindungsabbruch informiert wird, und gegebenenfalls andere notwendige Operationen getätigt werden (z.B. Entfernung aus List an Usern, die online sind) Oder auch im Code:

```
121         case "?LOGOUT&BUY" :
122           {
123             if (temp.getMyaccount()!=null) {
124               List<Artikel> plis = temp.getMyaccount().getBasket();
125               plis.toFirst();
126               while (plis.hasAccess()) {
127                 if(plis.getContent()!=null) this.myorders.bestellen(plis.getContent(),temp.getMyaccount());
128                 plis.next();
129               } // end of while
130               temp.getMyaccount().clearbasket();
131               this.send(pClientIP,pClientPort,"!LOGOUT&BUY:SUCCESS");
132               this.closeConnection(pClientIP,pClientPort);
133             } else {
134               this.send(pClientIP,pClientPort,"!LOGOUT&BUY:SUCCESS");
135               this.closeConnection(pClientIP,pClientPort);
136             } // end of if-else
137           }
138           break;
```

Alternativ kann der Server auch ohne Aufforderung die Verbindung mit dem bzw. allen Clients terminieren. Hierbei wird jedoch trotzdem der Client über diese Terminierung informiert.

```
236   public void processClosingConnection(String pClientIP, int pClientPort){
237     this.onlineusers.toFirst();
238     while (onlineusers.hasAccess()) {
239       if (onlineusers.getContent().getIp().equals(pClientIP)&&onlineusers.getContent().getPort()==pClientPort) {
240         this.onlineusers.remove();
241       } else {
242         this.onlineusers.next();
243       } // end of if-elsef
244     } // end of while
245     this.send(pClientIP,pClientPort,"CONNECTION_TERMINATED");
246   }
```