Praktikum 4 zur OOS SS16 G1: 03.06. & G2: 27.05.

Ziel:

In diesem Versuch wird das Zusammenspiel aller bisher erzeugten Klassen so implementiert, dass sowohl eine lokale Benutzerverwaltung, als auch eine remote-Benutzerverwaltung verwendet werden.

In Aufgabe 1 steht die Steuerung eines Systems aus mehreren Objekten bei lokaler Benutzerverwaltung im Vordergrund.

Die Verteilung des Systems in einen Client und einen remote-Server wird in Aufgabe 2 realisiert.

Hinweis:

Da dieser Praktikumsversuch naturgemäß der umfangreichste ist, ist eine intensive Vorbereitung unbedingt erforderlich.

Hierzu ist auch die Bearbeitung der Übungen sehr zu empfehlen!

Da die Systeme kompatibel zu denen anderer Studenten sein sollen, ist die Exception, die bei benutzerEintragen geworfen werden kann, als BenutzerSchonVorhandenException zu bezeichnen.

Aufgabe 1 (Steuerung mit lokaler Nutzerverwaltung):

In dieser Aufgabe wird das Zusammenspiel aller in den vorherigen beiden Versuchen erzeugten Klassen gesteuert. Dabei wird jedoch die `BenutzerVerwaltungAdmin` nur lokal verwendet. D.h. insbesondere, dass der Zustand der `RemoteCheckBox` momentan noch bedeutungslos ist. Das Zusammenspiel aller Objekte wird durch ein Steuerungsobjekt der Klasse `Client` gesteuert. Diese Klasse wird im Vorgriff auf Aufgabe 2 absichtlich nicht "Steuerung" genannt.

Legen Sie ein neues Paket an, in das Sie alle benötigten Dateien integrieren.

Implementieren Sie zusätzlich eine neue Klasse `Client`, die das Zusammenspiel aller Objekte steuert. Gehen Sie hierzu wie folgt vor:

Da die Klasse `Client` insbesondere die Erzeugung und Kommunikation mit verschiedenen Fenstern übernimmt, muss sie diesen Call-Back-Methoden anbieten. D.h., dass sie nicht nur aus einem reinen `main`-Programm bestehen darf, sondern in ihrem `main`-Programm lediglich ein Objekt der Klasse `Client` erzeugt wird und folgende Aufgaben in ihrem Konstruktor ausgeführt werden:

1. Erzeugung eines Objekts der Klasse `BenutzerVerwaltungAdmin`.
2. Anfrage an den Benutzer, ob die Datenhaltung initialisiert werden soll und ggf. Initialisierung. Verwenden Sie hierzu folgenden Code, um einen `int`-Wert von der Standardeingabe entgegenzunehmen:
```
BufferedReader din = new BufferedReader(
                         new InputStreamReader(System.in));
int dbInitialisieren = Integer.parseInt(din.readLine());
```
Damit steht der Eingabewert in der Variable `dbInitialisieren`.
3. Erzeugung eines `LoginFrames` mit Übergabe der eigenen Referenz für call-back. (Änderung des Konstruktors von `LoginFrame`.)

Dadurch wird die Kontrolle an den `LoginFrame` übergeben.

Praktikum 4 zur OOS SS16 G1: 03.06. & G2: 27.05.

(Auf die Spezifikation der call-back-Methoden von `Client` *wird immer dann eingegangen, wenn dadurch wieder die Kontrolle an* `Client` *gegeben wird.)*

Die Implementierung der Klasse `LoginFrame` wird wie folgt erweitert:

Als Reaktion auf Drücken des `jButtons` werden in Abhängigkeit des Attributs `neuAnmeldung` die beiden folgenden Fälle unterschieden:

Fall 1 `neuAnmeldung == true`:

Dann muss dem Nutzer ein `AnmeldungsFrame` zur Verfügung gestellt werden, damit er sich neu anmelden kann.

Hierzu wird die Kontrolle wieder an das `Client`objekt übergeben durch den Aufruf der Call-Back-Methode `neuAnmeldung()`, die in `Client` wie folgt zu implementieren ist:

Es wird lediglich ein `AnmeldungsFrame` mit Übergabe der eigenen Referenz für call-back erzeugt. (Änderung des Konstruktors von `AnmeldungsFrame`)

Dadurch wird die Kontrolle an `AnmeldungsFrame` übergeben.

Die Implementierung der Klasse `AnmeldungsFrame` wird wie folgt erweitert:

Als Reaktion auf Drücken des `jButtons` wird in dem Fall, dass die Inhalte der beiden Passwortfelder verschieden sind, verfahren wie bisher.

In dem Fall, dass sie gleich sind, wird die Kontrolle wieder an das `Client`objekt übergeben durch den Aufruf der Call-Back-Methode `neuerBenutzer(benutzer)`. Dabei wird im Parameter der in `AnmeldungsFrame` erzeugte Benutzer übergeben.

Die Call-Back-Methode `neuerBenutzer(Benutzer benutzer)` ist in `Client` wie folgt zu implementieren:

Es wird versucht, den neuen Benutzer in die `BenutzerVerwaltungAdmin` einzutragen durch Aufruf der Methode `benutzerEintragen(benutzer)`.

Ist das erfolgreich, wird dem Anwender angezeigt, dass er jetzt den Login-Vorgang durchführen kann, indem ein `LoginFrame` mit Übergabe der eigenen Referenz erzeugt wird. Für diesen Fall ist `LoginFrame` nicht weiter zu erweitern.

Beim Auftreten der Exception, wird ein neuer `AnmeldungsFrame` erzeugt und eine aussagekräftige Fehlermeldung in dessen oberstes Textfeld geschrieben. Hierzu ist dieses Textfeld als friendly zu deklarieren. Der Benutzer hat nun die Möglichkeit, sich mit anderer ID oder Passwort einzutragen.

Fall 2 `neuAnmeldung == false`:

In diesem Fall muss überprüft werden, ob der Benutzer sich bereits früher als Nutzer des Systems angemeldet hat. Hierzu wird die Kontrolle wieder an das `Client`objekt übergeben durch den Aufruf der Call-Back-Methode `benutzerLogin(benutzer)`. Dabei wird im Parameter der in `LoginFrame` erzeugte Benutzer übergeben.

Die Call-Back-Methode `benutzerLogin(Benutzer benutzer)` ist in `Client` wie folgt zu implementieren:

Praktikum 4 zur OOS SS16 G1: 03.06. & G2: 27.05.

Es wird durch Aufruf der Methode `benutzerOk(benutzer)` überprüft, ob der Benutzer bereits in `BenutzerVerwaltungAdmin` eingetragen ist.

Ist das der Fall, wird dem Anwender angezeigt, dass er nun das System benutzen kann, indem ein `AnwendungsFrame` mit Übergabe der eigenen Referenz erzeugt wird. Wegen der Einfachheit des Systems kann er aber nur noch den `jButton` drücken.

Im anderen Fall, dass `benutzerOk(benutzer) false` liefert, werden ein neuer `LoginFrame` erzeugt und eine aussagekräftige Fehlermeldung in dessen oberstes Textfeld geschrieben. Hierzu ist dieses Textfeld als friendly zu deklarieren. Der Benutzer hat nun die Möglichkeit, sich mit anderer ID oder Passwort anzumelden.

Aufgabe 2 (Netzverteilung):

In dieser Aufgabe wird zusätzlich zu der Verwendung einer lokalen Benutzerverwaltung aus Aufgabe 1 auch der Zugriff zu einer Remote-Benutzerverwaltung implementiert.

Hierzu wird auf lokaler Clientseite, neben der Implementierung des Interfaces durch `BenutzerVerwaltungAdmin` noch eine zweite Implementierung realisiert, die den Zugriff auf die Remote-Benutzerverwaltung realisiert.

Auf Serverseite wird lediglich eine Benutzerverwaltung zur Verfügung gestellt, die auf ankommende Anfragen des Clients wartet und diese dann bearbeitet.

Ändern Sie die Implementierung aus Aufgabe 1 so ab, dass sie in Client und Server aufgeteilt wird, die über eine Socketverbindung miteinander kommunizieren.

Legen Sie hierzu die folgenden 3 Pakete an:

a) **`prak4gemklassen`**
Dieses Paket enthält alle Klassen, die zwischen dem Client und dem Server transportiert werden. D.h., es muss auf jeden Fall die Klasse `Benutzer` und eine Exceptionklasse enthalten. Es könnte auch weitere gemeinsame Klassen von Client und Server enthalten. Hiervon sehen wir jedoch ab und legen jeweils Kopien in beide der folgenden Pakete an. Dieses Paket ist in den folgenden beiden Paketen zu importieren.

b) **`prak4client`**
Dieses Paket enthält neben den Klassen der Versuche 2 und 3 (ohne die Exception-Klasse und die Klasse `Benutzer`) die folgenden beiden Klassen:
- **`Client`**, die unter Verwendung der Klasse `ClientOrb` neben der Funktionalität aus Aufgabe 1 auch die Verwendung der remote-Benutzerverwaltung realisiert.
- **`ClientOrb`**, die die Verbindung zum Server über ein Objekt der Klasse `ServerOrb` realisiert und dem Client als Proxy (Ersatz) für die implementierende Klasse des Interfaces `BenutzerVerwaltung` im Remote-Fall dient.

c) **`prak4serv`**
Dieses Paket enthält neben den Klassen der Versuche 3 und 4 (ohne

Praktikum 4 zur OOS SS16 **G1: 03.06. & G2: 27.05.**

die Frame- und Exception-Klassen und die Klasse `Benutzer`) die folgenden beiden Klassen:
- **`Server`**, dient zur Steuerung auf Serverseite, d.h. es erzeugt ein Objekt `bv` der Klasse `BenutzerVerwaltungAdmin` und ein Objekt `so` der Klasse `ServerOrb`. Da `so` die Aufrufe von Diensten an `bv` delegiert, wird `so` die Referenz auf `bv` im Konstruktor übergeben und somit die Steuerung an `so` übergeben.
- **`ServerOrb`**, nimmt die Dienstanforderungen von `ClientOrb` mittels der Socketverbindung entgegen und liefert das Ergebnis, indem es die Dienstanforderungen durch Delegation an `mv` weiterleitet.

d) Führen Sie ggf. Casting-Operationen zur Transformation in die richtigen Datentypen durch.

e) Implementieren Sie eine sinnvolle Übertragung von Exceptions.

Zur Erweiterung der call-back Methoden des Clients ist folgendes zu beachten:

f) Das Szenario für eine remote-Neuanmeldung wird wie folgt festgelegt:
Da zuerst der `LoginFrame` erscheint und dieser der einzige ist, bei dem man auswählen kann, ob man die remote- oder die lokale Benutzerverwaltung verwenden möchte, muss schon im `LoginFrame` ausgewählt werden, ob man sich remote neuanmelden möchte oder nicht. Wenn ja, wird die IP des remote-Rechners im Client gespeichert und nach erfolgreichem Eintrag im `AnmeldungsFrame` der Benutzer remote eingetragen.

Teilen Sie zur Bearbeitung der beiden möglichen Neuanmeldungsfälle die Client-Methode `neuAnmeldung` in `neuAnmeldungLokal` und `neuAnmeldungRemote` auf, die ein neu zu definierendes `private`-Attribut `lokal` entsprechend setzen und in das neu zu definierende `private`-Attribut `address` ggf. die Remote-Adresse für die Neuanmeldung eintragen.
Beide Methoden erzeugen danach einen `AnmeldungsFrame`. Dieser `AnmeldungsFrame` ruft nach Drücken des Button die Clientmethode `neuerBenutzer` auf, die nicht zerlegt werden muss, aber nun in Abhängigkeit des Attributs `lokal` die Methode `benutzerEintragen` lokal oder remote aufruft.

g) Da nach der erfolgreichen Neuanmeldung wieder ein `LoginFrame` erscheint, bei dem wieder die remote-Adresse eingetragen werden muss, die sich von der im vorherigen `LoginFrame` unterscheiden kann, muss auch die Client-Methode `benutzerLogin` in zwei Methoden `benutzerLoginLokal` und `benutzerLoginRemote` zerlegt werden, die vom `LoginFrame` abhängig von dem Zustand der `JCheckBox` „lokal" aufgerufen werden. Diese Methoden rufen dann die Methode `benutzerOk` entweder lokal oder remote auf.

Achtung:

Wenn Sie die Tests für die Remoteverbindung lokal durchführen, müssen Sie beachten, dass die lokale und die remote-BenutzerVerwaltungen nicht in der gleichen Datei persistent gemacht werden, da es sonst zu Konflikten kommen kann.