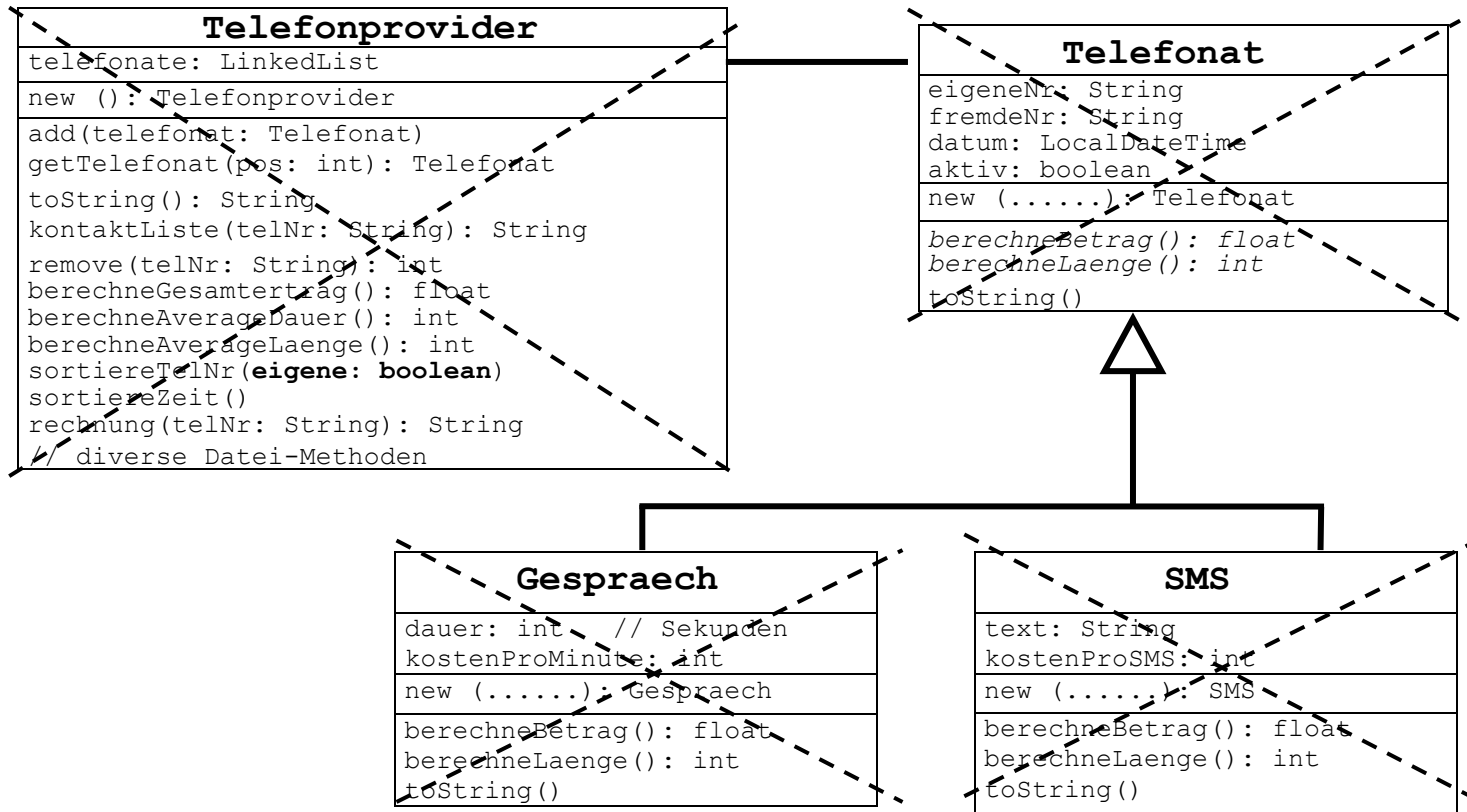


!! !!!!!!!!!!! Ihr Projektname in Eclipse: Ihr Nachname !!!!!!!!!!! !!

Der neu am Mobilmarkt auftretende Provider "TurboTel" benötigt zur Verwaltung seiner Kunden und deren Telefonate eine neuen Applikation.

Implementieren Sie als ein paar der dafür benötigten Module sowie zum Testen derselben die im nachfolgenden UML-Diagramm beschriebenen Klassen:



Die **get**- und **set**-Methoden der jeweiligen Klassen sowie die Konstruktoren sind im Diagramm nicht dargestellt, aber obligatorisch.

Aufgabe1: TelefonException zur Behandlung sämtlicher Fehler-Situationen.

Aufgabe2: Telefonat

Konstruktor übernimmt Werte für alle Attribute.
getter/setter Sowohl **eigeneNr** als auch **fremdeNr** bestehen nur aus Ziffern; beginnen die Nummern mit "0699" haben sie 12 Stellen, sonst 11.
berechneBetrag() hat hier keine Implementierung; muss in allen ableitenden Klassen implementiert werden.
getLaengeAsString() hat hier keine Implementierung; muss in allen ableitenden Klassen implementiert werden.
toString() "eigeneNr: " eigeneNr " fremdeNr: " fremdeNr " Zeit: " Datum u. Uhrzeit

Beispiel: eigeneNr: 06761234567 fremdeNr: 069912345678 Zeit: 2009-6-24, 13:55

Aufgabe3: Gespraech

Konstruktor übernimmt zus. Werte für **dauer** (in Sekunden, zw. 1 und 6000) und **kostenProMinute** (zw. 1 Cent und 50 Cent).
berechneBetrag() Berechnung erfolgt in 60-Sekunden-Taktung (z.B. Dauer 61 Sek. -> Abrechnung 2 Minuten!) und nur für ausgehende (=aktiv) Anrufe!
getLaengeAsString() liefert die Dauer des Gesprächs in folgender Form: dauer "Sekunden"
toString() liefert zusätzlich die Dauer und die Kosten dieses Gesprächs.

Beispiel: eigeneNr: 06761234567 fremdeNr: 069912345678 Zeit: 2009-06-24, 13:55, 90 Sekunden -> 7.5 Cent

Aufgabe3b: TestGespraech

Aufgabe4: SMS

Konstruktor

übernimmt zus. den Text der SMS sowie die **kostenProSMS** (zwischen 1 Cent und 20 Cent).

berechneBetrag()

Eine normale SMS besteht aus 0 (= leere SMS!) bis 160 Zeichen; hat sie mehr Zeichen (überlange SMS), sind für diese überlange SMS entsprechend viele SMS-Teile zu je(!!) 153 Zeichen zu kalkulieren!! Verrechnet werden nur ausgehende (=aktive) SMS !!!!!!!!!!!

Beispiele: 160 Zeichen -> = 1 SMS; 161 Zeichen -> = 2 SMS; !!!
306 Zeichen -> = 2 SMS; 307 Zeichen -> 3 SMS; usw...

ACHTUNG!!! 0 Zeichen -> = trotzdem 1 SMS !!!!!!!!!!!

getLaengeAsString()

liefert die Länge des Textes folgender Form: text-Länge "Zeichen"

toString()

liefert zusätzlich die Länge und die Kosten dieser SMS.

Beispiel: eigeneNr: 06761234567 fremdeNr: 069912345678 Zeit: 2007-3-24, 13:55, 480 Zeichen -> 60 Cent

Aufgabe4b: TestSMS

Aufgabe5: Telefonprovider

Konstruktor

legt die Collection an.

add(telefonat)

nimmt das Telefonat in die Collection auf.

toString()

Form: "Telefon-Provider TurboTel"

Infos über sämtliche Telefonate.....

kontaktListe(telNr)

liefert eine Aufstellung über sämtliche Telefon-Nummern, mit der die **telNr** Kontakt hatte, in folgender Form:

Beispiel:

Rufnummern-Liste für 069987654321

06501717171 -> aktiv
069919876543 -> aktiv
06765776161 <- passiv
06509876543 -> aktiv

remove(telNr)

löscht alle **Telefonate** aus d. Collection, die der übergebenen Nummer entsprechen und gibt die Anzahl der entfernten Nummern zurück.

berechneGesamtertrag()

berechnet d. Gesamt-Ertrag aus sämtlichen **Telefonaten**.

berechneAverageDauer()

berechnet d. durchschnittliche Gesprächs-Dauer aller **Gespraeche!**

berechneAverageLaenge()

berechnet d. durchschnittliche Länge aller **SMS!**

sortiereTelNr(aktiv)

sortiert d. Liste mit Hilfe d. **Collections.sort**-Methode u. Comparator aufsteigend nach **eigeneNr** oder **fremdeNr** (je nachdem, ob d. Parameter **aktiv true** oder **false** ist).

sortiereZeit()

sortiert d. Liste mit Hilfe d. **Collections.sort**-Methode absteigend nach **zeit !!**.

rechnung(telNr)

liefert eine Aufstellung sämtlicher abgegangener **Telefonate** dieser Tel.Nummer (nach Zeit sortiert!) in folgender Form:

Telefon-Abrechnung für Rufnummer 069987654321

Infos über alle Telefonate dieser Nummer !!!!!!!!!!!

Gesamt-Betrag:

Aufgabe5b: TestProvider

zum laufenden Testen der Methoden

Aufgabe6: **saveTelefonate(filename)**
speichert die Daten aller **Telefonate** in einer serialisierten Datei beliebigen Namens (z.B. "telefonate.ser") auf "c:\scratch".

Aufgabe6b: TestSave

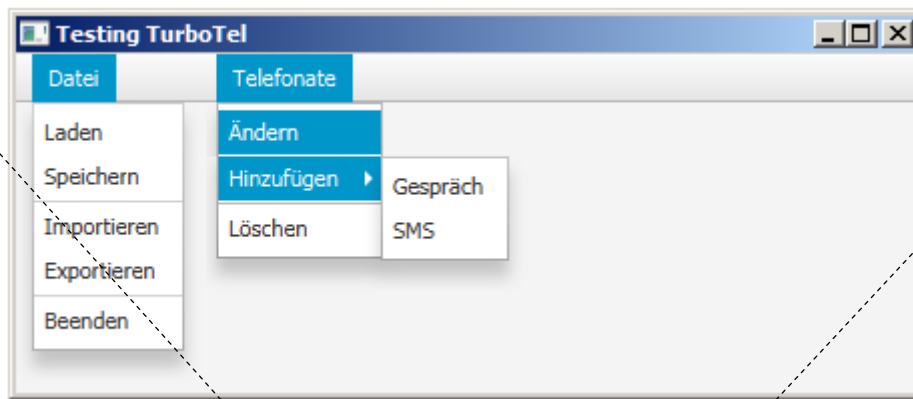
!!! ACHTUNG !!! Wenn hierbei eine **IOException** auftritt, deren Ursache Sie auch mit Hilfe d. Debuggers(!) nicht finden können, wenden Sie sich an Hr. Schmid!!

Aufgabe7: **loadTelefonate(filename)**
lädt die Telefonat-Daten aus einer serialisierten Datei beliebigen Namens und fügt die Telefonate der bestehenden Collection hinzu.

Aufgabe7b: TestLoad

Aufgabe8 (GUI): ACHTUNG! (Fehler-)Meldungen sind nur über **graphische** Werkzeuge und nicht etwa auf der Systemkonsole auszugeben!!

Erstellen Sie für das Testen der neuen Abrechnungs-Software eine graphische Oberfläche (Größe: 450*250 Pixel), die folgendes Aussehen und Menü-Struktur aufweist (aber vorerst noch ohne jegliche Funktionalität!!):



Aufgabe9:

Jene Menüpunkte, die ohne geladene Dateien nicht sinnvoll betätigt werden können, sollen nicht "aktivierbar" sein.

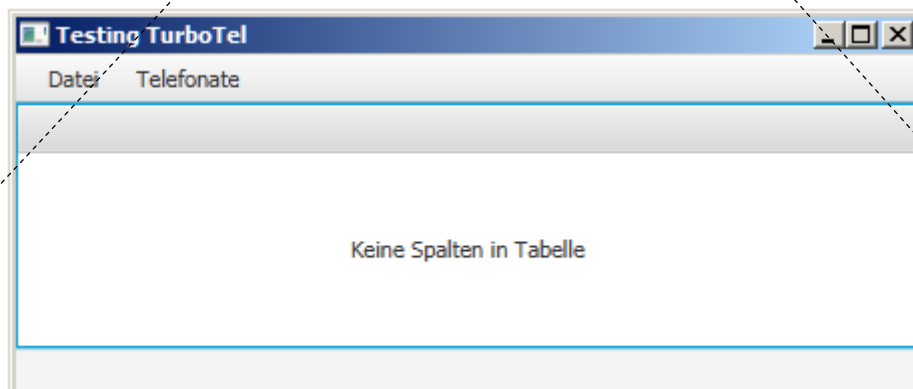
Aufgabe10:

Das Anklicken der Menüpunkte "**Beenden**", "**Laden**" und "**Speichern**" soll zu entsprechenden Aktivitäten führen.
Sowohl beim Laden als auch beim Speichern sollen die entsprechenden Dateien bzw. Speicherorte mit Hilfe eines Datei-Dialogs ausgewählt werden können, der automatisch den Ordner "**c:\scratch**" zur Datei-Auswahl anbietet.
Beim "Laden" sind die Daten noch nicht(!) tabellarisch darzustellen; die korrekte Laden-Funktionalität kann aber prinzipiell über ein S.o.p.(...) überprüft werden oder durch das "Speichern" und einen kurzen Vergleich der Dateien im Ordner c:\scratch.

Aufgabe11:

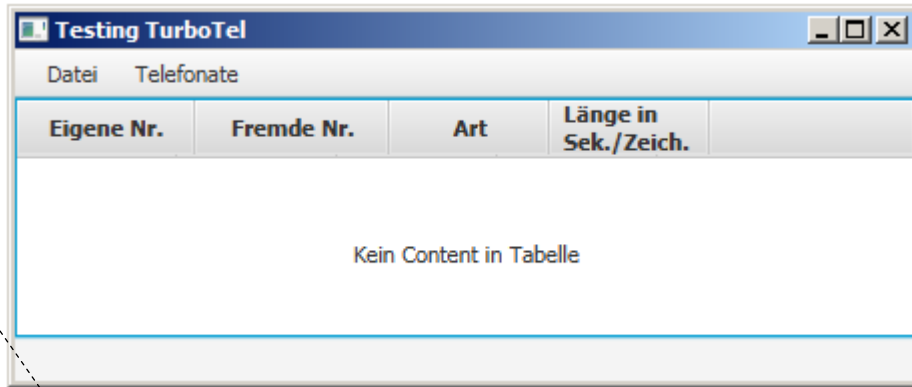
Implementieren Sie für die Darstellung der **Telefonate**-Daten ein Übersichts-Fenster; es soll später erst nach dem Laden(!) von Daten sichtbar werden!!

Aussehen bei einem sofortigen ersten Test des leeren Übersichts-Fensters aus der **laden()**-Methode heraus, wenn noch **keinerlei** Spalten, Methoden etc. implementiert wurden:



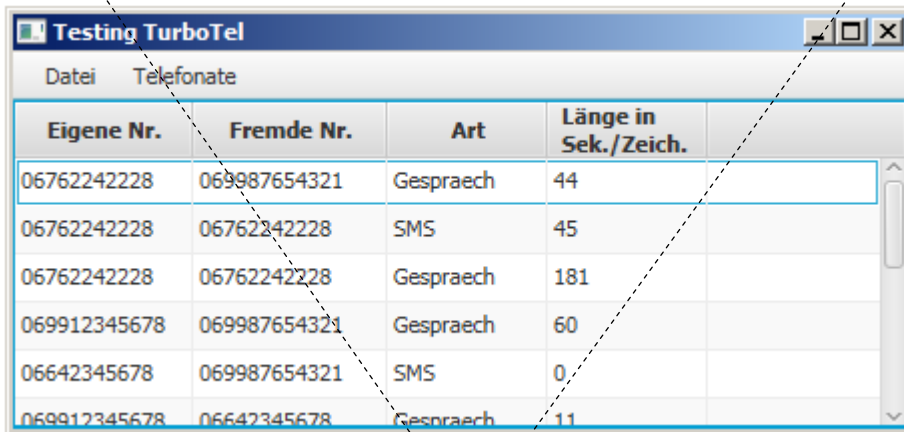
...oder...

...oder, wenn nur die Spalten implementiert wurden, aber noch keine Daten-Aufbereitung:



Eigene Nr.	Fremde Nr.	Art	Länge in Sek./Zeich.
Kein Content in Tabelle			

...bzw. nach dem Laden und vollständiger Daten-Aufbereitung:

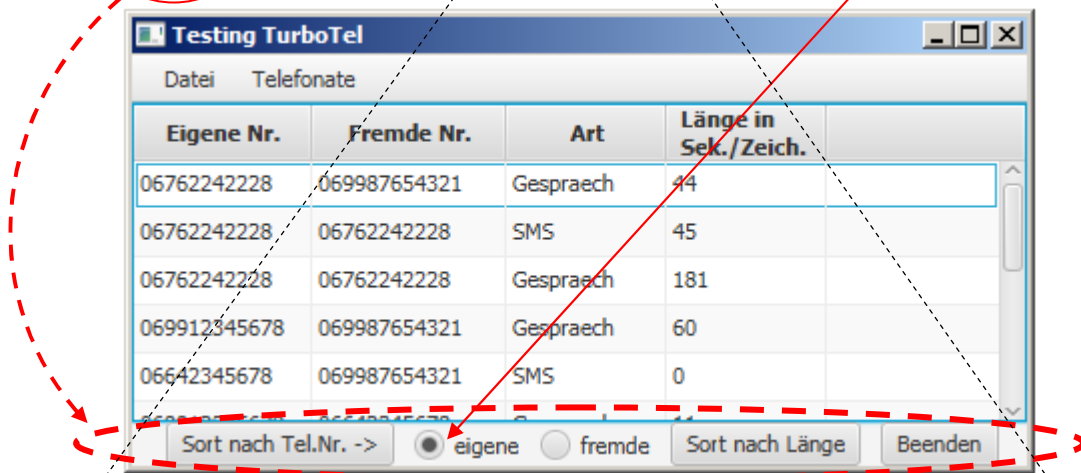


Eigene Nr.	Fremde Nr.	Art	Länge in Sek./Zeich.
06762242228	069987654321	Gespraech	44
06762242228	06762242228	SMS	45
06762242228	06762242228	Gespraech	181
069912345678	069987654321	Gespraech	60
06642345678	069987654321	SMS	0
069912345678	06642345678	Gespraech	11



HINWEIS: Verwenden Sie für die "Länge"-Information die in der **Telefonat**-Klasse implementierte Methode **getLaengeAsString()**.

Aufgabe12: Erweitern Sie die das Hauptfenster wie folgend beschrieben; die Buttons sollen noch keinerlei Funktion haben, "**eigene**" soll aber bereits "ausgewählt" sein! Auch diese Komponenten sollen erst nach dem Laden von Daten sichtbar werden!!!!



Eigene Nr.	Fremde Nr.	Art	Länge in Sek./Zeich.
06762242228	069987654321	Gespraech	44
06762242228	06762242228	SMS	45
06762242228	06762242228	Gespraech	181
069912345678	069987654321	Gespraech	60
06642345678	069987654321	SMS	0
069912345678	06642345678	Gespraech	11

Sort nach Tel.Nr. -> ☒ eigene ☐ fremde Sort nach Länge Beenden

Aufgabe13: Stattdessen Sie nun die Buttons im unteren Fenster-Teil mit der entsprechenden Funktionalität aus; die dafür benötigten Sort-Methoden bzw. Comparatoren sind in der Verwaltungs-Klasse zu implementieren (siehe **Aufgabe 5**)!

Start der Aufgaben für die 2. PLÜP

Aufgabe14:

Bei Markierung **mehrerer** Telefonate in der Übersicht und Auswahl des Menüpunktes "**Telefonate -> Löschen**" sind die markierten Telefonate aus der Collection zu entfernen.

Aufgaben15ff (SmsDialog):

WICHTIG!!

Beginnen Sie mit der Implementierung dieser Dialog-Klasse zuerst in der besprochenen und geübten Weise.

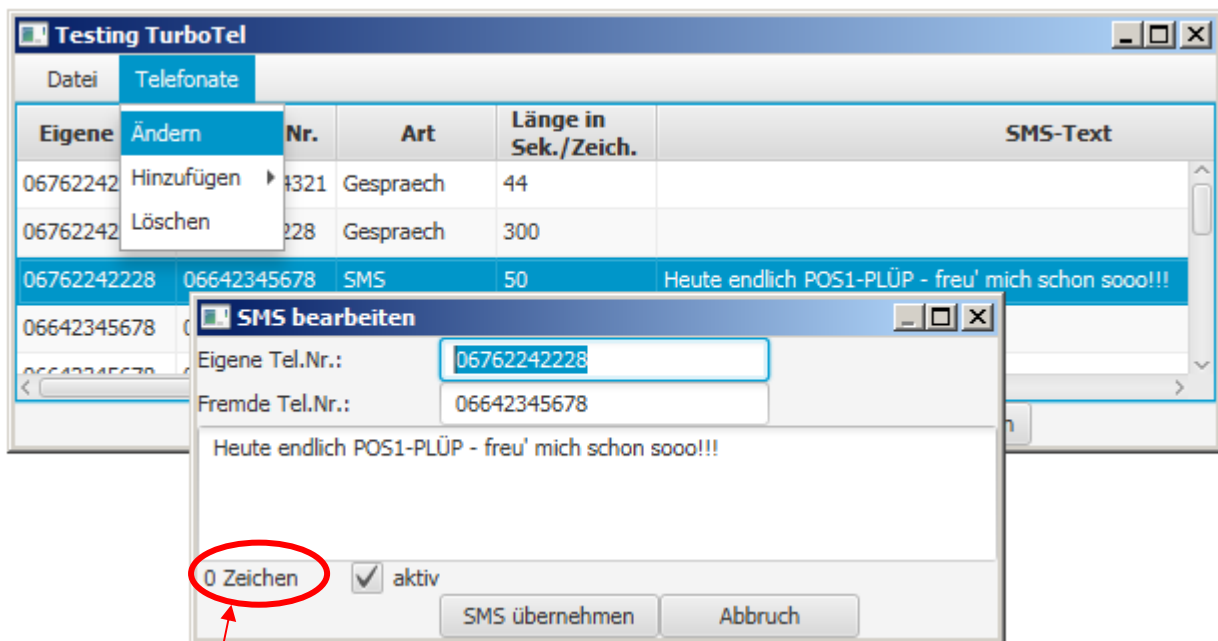
Unterbrechen Sie nach dem Anlegen der Klasse die Implementierung aber, bevor Sie mit der Deklaration der Attribute beginnen(!), und kopieren Sie aus der beiliegenden Rumpf-Klasse

"CodeVorgabe_fuer_SmsDialog.java"

zuerst die Zeilen mit den **import**-Statements an den Anfang ihrer Klasse (direkt **UNTER** die Package-Anweisung!!), sowie die drei nachfolgenden "Code-Vorgabe - Blöcke" **IN(!)** die Klasse hinein.

Aufgabe15 (Dialog "Ändern"):

Bei Auswahl einer markierten **SMS** und des Menüpunktes "**Telefonate -> Ändern**" soll folgendes Dialogfenster erscheinen (Klasse **SmsDialog**, Größe 400 x 150):

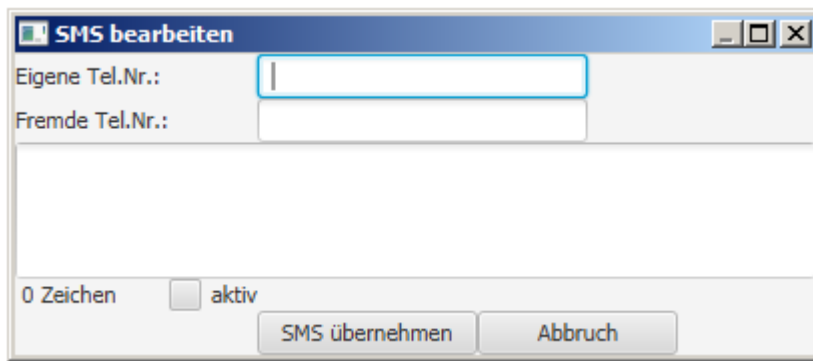


Während der Dialog sichtbar ist, darf im Hintergrund nicht weitergearbeitet werden können!

ACHTUNG!! Dieses Label hat noch keinerlei Funktionalität!!

Aufgabe16 (Dialog "SMS aufnehmen"):

Bei Auswahl des Menüpunktes **"Telefonate -> Hinzufügen -> SMS"** soll mit Hilfe des soeben implementierten Dialogs (und möglichst wenig ergänzendem Code) die Neu-Aufnahme einer SMS ermöglicht werden; verwenden Sie dafür den Standard-Konstruktor der Klasse SMS !



Aufgabe17:

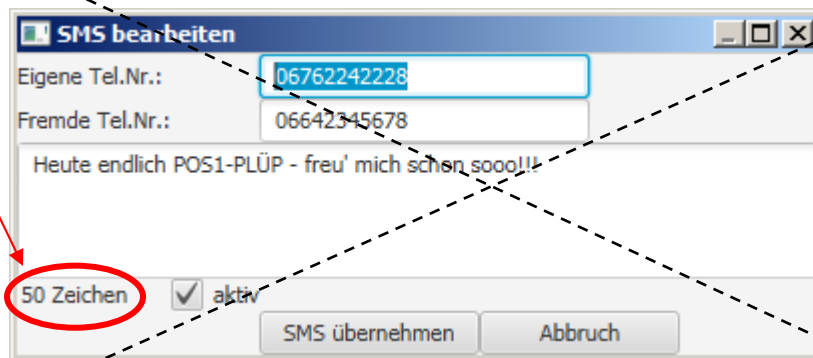
Auch durch einen Doppelklick auf eine Tabellen-Zeile soll das **"Bearbeiten"** einer SMS aufgerufen werden können.

Aufgabe18 (Bonus):

Bei einem Rechtsklick auf eine/mehrere Tabellen-Zeile(n) soll das **"Ändern"** / **"Löschen"** mittels Context-Menü möglich sein.

Aufgabe19 (Bonus): ...aber nur für jene, denen schon echt fad ist!! ;o)

Die Anzahl der eingetippten Zeichen ist im Dialog laufend zu aktualisieren; d.h.: bei jedem getippten Zeichen soll sofort die aktuelle Anzahl erscheinen!!



(**Hinweis:** Sie müssen sich dafür mittels der Methode `taText.textProperty()` Zugriff auf diese verschaffen und dort einen Listener hinzufügen (!!); verwenden Sie dafür die Lambda-Schreibweise!)