**Programmierung II – Übung 1**

## Projekt Adressbuch-V1T

Öffnen Sie mit *File->Open Project* in *Netbeans* das Projekt *Adressbuch_VT1* aus dem Ordner *UBG_1*. In diesem Projekt wird – ähnlich wie in der Vorlesung – die Klasse *Adressbuch* und die Klasse *Kontakt* als Dienstleister genutzt. Hier wird das Adressbuch jedoch nicht über eine grafische Schnittstelle bedient, sondern über eine textuelle Schnittstelle.

Probieren Sie diese einmal aus, indem Sie *AdressbuchDemo* starten. Die Klasse hat eine main-Methode, in der die Textschnittstelle des Adressbuchs aufgerufen wird. In Netbeans erkennen Sie Java-Klassen mit einer main-Methode daran, dass diese ein kleines grünes Dreieck besitzen. Sie können diese Klassen starten, indem Sie die Datei selektieren und im Kontextmenü *Run File* aufrufen.

Das Adressbuch lässt sich in der Konsole (unter dem Editor auf der rechten Seite) über textuell eingegebene Befehlsworte bedienen. Beginnen Sie mit dem Wort „hilfe“ um sich eine Liste der gültigen Befehle anzeigen zu lassen und probieren Sie die Befehle aus.

Versuchen Sie nun herauszufinden, wie die textuelle Schnittstelle funktioniert. Sehen Sie sich das Zusammenwirken der Klassen *AdressbuchDemo*, *AdressbuchTexteingabe*, *Parser* und *Befehlswoerter* genau an. Versuchen Sie zu verstehen, was die Klasse *Scanner* macht.

Sehen Sie sich danach die Klassen *Adressbuch* und *Kontakt* an. Diese sind genau so, wie ich sie in der Vorlesung vorgestellt habe.

## Erweitern der Benutzerschnittstelle

Die textuelle Schnittstelle macht keinen Gebrauch von den Methoden *getKontakt(..)* und *deleteKontakt(..)*. Momentan kann man nur per Präfix nach Kontakten suchen und keinen Kontakt wieder aus dem Adressbuch löschen.

Erweitern Sie die Klassen *Befehlswoerter* und *AdressbuchTexteingabe* so, dass auch ein interaktiver Zugriff auf diese Methoden möglich ist. Hierzu müssen Sie zunächst den Satz der Befehlswörter um die Worte „hole“ und „entferne“ erweitern.

Die Klasse *AdressbuchTexteingabe* muss dahingehend erweitert werden, dass auf die Eingabe dieser Befehlswörter entsprechend reagiert werden kann. Legen Sie dazu in der Klasse die folgenden Methoden an:

```
private void holeEintrag()

private void entferneEintrag()
```

Sie können sich hierzu sehr gut an dem Umgang mit dem Befehl *suche* mit Hilfe der Methode *sucheEintrag()* orientieren. Beim Holen des Eintrags sollte nur der Kontakt zu

dem eingegebenen Schlüssel geliefert werden, nicht die Kontakte, die den Schlüssel als Präfix haben.

Erweitern Sie nun die textuelle Schnittstelle dahingehend, dass ein bereits eingetragener Kontakt auch geändert werden kann. Die Klasse *Adressbuch* bietet dazu die Methode *updateKontakt(..)* an. Hier muss der Schlüssel des *Kontakt*-Objekts angegeben werden, das ersetzt werden und das neue *Kontakt*-Objekt, das anstelle des alten Objekts gespeichert werden soll.

Ändern Sie in der Klasse *AdressbuchTexteingabe* hierzu zunächst die Methode *neuerEintrag()*. Hier werden die einzelnen Attributwerte für ein *Kontakt*-Objekt eingelesen und daraus ein neues *Kontakt*-Objekt erzeugt. Da Sie dieselbe Funktionalität auch für das Ändern eines Kontakts benötigen, sollten Sie diese in eine eigene Methode auslagern. Fügen Sie der Klasse *AdressbuchTexteingabe* also eine Methode

```
private Kontakt kontaktEinlesen()
```

hinzu, in der die einzelnen Attributwerte eines Kontakts eingelesen und ein neues *Kontakt*-Objekt erstellt wird, das die Methode an den Aufrufer zurück liefert.

Nutzen Sie diese Methode in der Methode *neuerEintrag()*, um das neu zu erstellende *Kontakt*-Objekt einzulesen. Dabei entfallen viele Anweisungen.

Schreiben Sie nun eine Methode `private void aendereEintrag()`, die in der Klasse *AdressbuchTexteingabe* verwendet werden kann, um einen existierenden Eintrag zu ändern.

**Objekterzeugung verhindern**

Eine wichtige Verwendung von Exceptions dient dazu, die Erzeugung von Objekten zu verhindern, die nicht in einen gültigen Anfangszustand versetzt werden können. Üblicherweise ist das der Fall, wenn einem Konstruktor ungültige Parameterwerte übergeben werden. Sehen Sie sich den Konstruktor der Klasse *Kontakt* an. Der Konstruktor weist momentan null-Werte bei den Parametern nicht zurück, sondern ersetzt diese durch den leeren String. Allerdings benötigt das Adressbuch mindestens einen gültigen Namen oder eine gültige Telefonnummer als eindeutigen Schlüssel.

Ändern Sie den Konstruktor dahingehend ab, dass er eine *IllegalStateException* wirft, wenn sowohl für den Namen als auch die Telefonnummer ein leerer String übergeben wurde. Geben Sie der Exception im Konstruktor einen entsprechenden Fehler-String mit. Informieren Sie sich in der API-Dokumentation, ob es sich bei dieser Exception um eine geprüfte oder ungeprüfte handelt.

**Tipp**: Eine ungeprüfte Exception ist immer von *RuntimeException* abgleitet.

In unserem Beispiel kann der Nutzer des Adressbuchs sehr wohl auf diesen Fehler reagieren. Das Erzeugen eines neuen Kontakts findet statt, wenn ein Kontakt in das Adressbuch aufgenommen oder ein bestehender Kontakt des Adressbuchs geändert

werden soll. Gibt der Nutzer falsche Eingabewerte vor, so kann dieser auf den Fehler aufmerksam gemacht und erneut zur Eingabe der Daten aufgefordert werden.

Bauen Sie die entsprechenden Try-Catch-Blöcke in die Methoden *neuerEintrag()* und *aendereEintrag()* ein. Geben sie bei der Behandlung der Exception zunächst die Fehlermeldung aus der Exception auf der Konsole aus und wiederholen Sie dann die Nutzerinteraktion.

**Tipp**: mit der Methode *getMessage()* können Sie die Fehlermeldung auslesen, die der Exception bei der Erzeugung mitgegeben wurde.

**Methodenparameter prüfen**

Lassen Sie erneut die Klasse *AdressbuchDemo* ablaufen. Versuchen Sie nun einen nicht-existenten Kontakt zu entfernen, indem Sie das Befehlswort „entferne“ eingeben und dann z.B. als Suchschlüssel „otto“ angeben. Hier kommt es zu einem Laufzeitfehler, einer *NullPointerException*. Diese entsteht, weil kein Kontakt zu dem angegebenen Schlüssel gefunden wird (die Objektreferenzvariable *kontakt* daher mit *null* belegt ist) und wir in der zweiten Zeile versuchen mit Hilfe des Punktoperators auf die Methode *getName()* zuzugreifen.

Der nicht abgefangene Laufzeitfehler führt im Programm zu einem sofortigen Absturz. D.h. wenn man bei dem Versuch, einen Kontakt zu entfernen, einen falschen Schlüssel eingibt, so stürzt das Programm ab. Ist das Ihres Erachtens ein akzeptables Verhalten?

Nein, natürlich ist ein solches Verhalten nicht wirklich akzeptabel. Als Nutzer einer Adressbuch-Software würde man erwarten, dass man darauf hingewiesen wird, dass für den angegebenen Schlüssel kein Eintrag existiert und das Programm dann wieder auf Nutzereingaben reagieren kann.

In erster Näherung könnte man nun auf die Idee kommen, die *NullPointerException* in der Methode *entferneEintrag()* der Klasse *AdressbuchTexteingabe* aufzufangen und dafür zu sorgen, dass im catch-Block eine entsprechende Fehlermeldung ausgegeben wird. Da jedoch die *NullPointerException* nicht von einer eigenen Methode geworfen wird, können wir der Exception keinen Fehlertext mitgeben, der den aufgetretenen Fehler genauer erklärt.

Darüber hinaus kann es auch zu einer *NullPointerException* kommen, wenn man die Methode *deleteKontakt(..)* in *Adressbuch* mit dem Wert *null* aufruft. Probieren Sie es aus, indem Sie in der *main*-Methode der Klasse *AdressbuchDemo* (vor Aufruf der textuellen Schnittstelle) auf dem Adressbuch im Attribut *buch* die Methode *deleteKontakt(..)* mit dem Wert *null* aufrufen. Hier kommt es zu einer *NullPointerException*, weil der angegebene Schlüssel ungültig ist, nicht weil kein passender Eintrag gefunden wurde.

Das Auftreten einer *NullPointerException* kann also verschiedene Gründe haben:

1. Für den angegebenen Schlüssel konnte kein Eintrag gefunden werden.
2. Das übergebene Argument war nicht gültig.

Der erste Fehler kann vom Nutzer des Programms korrigiert werden indem er einen anderen Schlüssel angibt. Der zweite Fehler kann nur vom Programmierer korrigiert werden, da es dem Nutzer des Programms gar nicht möglich ist, bei diesem Aufruf den Wert *null* einzugeben. Der Wert *null* muss also durch einen Fehler des Programmierers entstanden sein.

Daher sollten die beiden Fehler sehr genau unterschieden werden. Der erste Fehler sollte dem Nutzer gemeldet werden, hierbei muss eine klare Fehlerbotschaft ausgegeben werden. Der zweite Fehler muss dem Programmierer gemeldet werden, indem man einen Programmabbruch herbeiführt. Auch dieser Fehler sollte eine klare Fehlerbotschaft mit sich führen.

Ändern Sie daher die Methode *deleteKontakt(..)* in *Adressbuch* folgendermaßen:

Falls der als Argument übergebene Schlüssel den Wert *null* haben sollte, so werfen Sie eine *IllegalArgumentException*. Geben Sie dem Konstruktor als Argument eine entsprechende Fehlermeldung mit.

Falls für den Schlüssel kein Kontakt gefunden werden konnte, werfen Sie eine *NullPointerException*. Geben Sie auch hier im Konstruktor eine klare Fehlermeldung mit. Erweitern Sie den Javadoc-Kommentar dahingehend, dass Sie die *NullPointerException* hier dokumentieren.

Probieren Sie aus, ob die beiden Fehler sich jetzt klarer einordnen lassen, indem Sie in der main-Methode der Klasse *AdressbuchDemo* auf dem Adressbuch im Attribut *buch* die Methode *deleteKontakt(..)* aufrufen und die zwei Varianten der falschen Eingabe testen. Die Fehlermeldungen sind jetzt deutlich differenzierter, da es sich um unterschiedliche Fehlerarten handelt und der Fehlertext mit ausgegeben wird.

**Wiederaufsetzen des Programms**

Nun können Sie in der Methode *entferneEintrag()* der Klasse *AdressbuchTexteingabe* einen Try-Catch-Block einfügen, in dem die *NullPointerException* der Methode *deleteKontakt(..)* in *Adressbuch* so abgefangen wird, dass der Nutzer informiert wird und das Programm danach auf weitere Befehle des Nutzers wartet.

**Zusatzaufgabe**

Prüfen Sie, ob in der Klasse *Adressbuch* auch in den anderen Methoden eine *IllegalArgumentException* angebracht wäre. Fügen Sie diese – wo notwendig – ein.

Sie können bei dieser Gelegenheit auch prüfen, ob es auch in anderen Methoden wegen eines nicht-vorhandenen Eintrags zu einem gültigen Schlüssel zu einer *NullPointerException* kommen kann.