Programmierung II – Übung 2

Öffnen Sie mit *File->Open Project* in *Netbeans* das Projekt *Adressbuch_VT2* aus dem Ordner *UBG_2*. Dieses Projekt enthält den Stand der Adressbuch-Anwendung, wie er nach Durchführung der Übung 1 (ohne Zusatzaufgabe) sein sollte. Arbeiten Sie mit dieser Version weiter. Sie können allerdings kurz vergleichen, ob Ihre Lösung ähnlich aussah. Zur Erinnerung, folgende Methoden wurden verändert:

AdressbuchTexteingabe: *starten(), neuerEintrag()*

AdressbuchTexteingabe, neu: *holeEintrag(), entferneEintrag(), kontaktEinlesen(), aendereEintrag()*

Adressbuch: Konstruktor, *deleteKontakt()*

Kontakt: Konstruktor

Prüfen der Methodenparameter

Bei einer Dienstleistungsklasse sollte in jeder Methode geprüft werden, ob die Methodenparameter gültige Werte enthalten, bevor man mit den Werten arbeitet. Bei dem Adressbuch muss sichergestellt werden, dass kein Parameter den Wert *null* besitzt.

In der Klasse *Adressbuch* kommen als Methodenparameter im Wesentlichen Werte vom Typ *String* (Schlüssel) und *Kontakt* vor. Sorgen Sie zunächst dafür, dass überall dort, wo Methodenparameter vom Typ *Kontakt* auftreten, geprüft wird ob der Parameter den Wert *null* hat. Ist dies der Fall, so soll die ungeprüfte *IllegalArgumentException* geworfen werden. Da diese nicht gefangen werden und zu einem Abbruch des Programms führen sollen, sollten Sie der Exception eine für den Programmierer aufschlussreiche Nachricht mitgeben.

Nutzen der Methode *schluesselBekannt()*

Die Prüfung des Schlüssels können wir in die Methode *schluesselBekannt()* der Klasse *Adressbuch* auslagern. Diese Methode prüft ob der angegebene Schlüssel im Adressbuch verwendet wird.

Setzen Sie diese Methode in jeder Methode der Klasse *Adressbuch* ein, der ein Schlüssel übergeben wird. So kann sie z.B. in der Methode *gibKontakt(..)* verwendet werden, um herauszufinden, ob es überhaupt einen Kontakt zu diesem Schlüssel gibt.

Bauen Sie die Methode *schluesselBekannt()* sinnvoll in jede Methode ein. Wichtig ist hier, dass die Methode aufgerufen wird, bevor man versucht, über den Schlüssel auf einen Kontakt zuzugreifen.

In der Methode *deleteKontakt(..)* hatten wir bereits eine Prüfung des übergebenen Schlüsselparameters eingebaut. Wenn der Schlüssel den Wert *null* hat, so wird eine *IllegalArgumentException* geworfen. Diese Prüfung muss natürlich auch in den anderen Methoden, denen ein Schlüssel übergeben wird, durchgeführt werden. Da jedoch alle die Methode *schluesselBekannt(..)* aufrufen, reicht es die Prüfung dort durchzuführen. Verschieben Sie daher die Prüfung des eingegebenen Schlüssels in die Methode *schluesselBekannt(..)*.

Testen Sie, ob sich das Programm noch korrekt verhält. Starten Sie *AdressbuchDemo*, indem Sie die Datei selektieren und im Kontextmenü *Run File* aufrufen. Versuchen Sie, einen Kontakt zu entfernen, den es nicht gibt (z.B. mit dem Schlüssel „otto“). Wird hier noch der Fehler gemeldet, dass es keinen Kontakt zu dem Schlüssel gibt (Message der *NullPointerException*)?

Wenn nicht, dann korrigieren Sie dies.

Geprüfte Exceptions

Bisher haben wir in der Dienstleisterklasse *Adressbuch* ungeprüfte Exceptions verwendet, um den Programmierer auf Fehler aufmerksam zu machen. Ungeprüfte Exceptions führen im Normalfall zu einem Programmabsturz, da sie auf einen logischen Fehler im Programm des Klienten hinweisen.

Es gibt jedoch im Zusammenhang mit den Werten der Methodenparameter Fehler, die durch eine falsche Eingabe des Nutzers zustande kommen. Diese können vom Nutzer korrigiert werden und sollten daher nicht zu einem Programmabbruch führen. Solche Fehler können als geprüfte Exceptions modelliert werden. Geprüfte Exceptions werden in der Dienstleisterklasse explizit deklariert und dokumentiert, der Programmierer des Klienten muss solche Exceptions in seinem Programm behandeln, so z.B. indem er dem Nutzer den Fehler meldet und ihn zu einer erneuten Eingabe der Werte auffordert.

Fügen Sie eine neue Klasse *UngueltigerSchluesselException* hinzu, die das Auftreten eines ungültigen Schlüssels (leerer String) modelliert.

Tipp: Sie können einem Paket einfach in *Netbeans* eine neue Klasse hinzufügen, indem Sie links im Project-Browser das Paket und im Kontextmenü *New->Java Class* auswählen. Im nachfolgenden Fenster können Sie den Namen der Klasse eingeben. Es wird automatisch eine leere Java-Klasse erzeugt.

Orientieren Sie sich beim Schreiben der Klasse an der in der Vorlesung vorgestellten Klasse.

Tipp: Sie können in *Netbeans* einer Java-Klasse einen Konstruktor hinzufügen, indem Sie den Kursor in den Rumpf der Klasse setzen und dort im Kontextmenü *Insert Code...* wählen. Im nachfolgenden Menü wählen Sie *Constructor ...*. Im nachfolgenden Fenster können Sie sich links eine Konstruktor-Variante auswählen und rechts auswählen, für welche Attribute der Klasse zusätzliche Methodenparameter erstellt werden sollen.

Sorgen Sie dafür, dass diese Exception immer dann geworfen wird, wenn ein ungültiger Schlüssel übergeben wurde. Überlegen Sie gut, wo Sie diesen Fehler werfen, wenn Sie mehr als eine throw-Anweisung einfügen, haben Sie etwas falsch gemacht ;-) Denken Sie auch daran, die Exception so zu kommentieren, dass diese in der Javadoc-Dokumentation aufgeführt wird.

Übersetzen Sie die Klasse *Adressbuch*. Sie werden feststellen, dass der Compiler bei geprüften Exceptions erwartet, dass man sie entweder behandelt, oder propagiert. Da in der Klasse *Adressbuch* das Auftreten eines ungültigen Schlüssels nicht sinnvoll behandelt werden kann, muss die Exception, überall wo sie auftritt, zum Klienten propagiert werden. Konsultieren Sie die Vorlesungsfolien, um herauszufinden, was zu tun ist.

Wenn sich die Klasse *Adressbuch* korrekt übersetzen lässt, dann haben Sie vermutlich alles richtig eingebaut. Die *UngueltigerSchluesselException* wird nun bis zu der aufrufenden Methode im Klienten *AdressbuchTexteingabe* propagiert.

Versuchen Sie nun die Klasse *AdressbuchTexteingabe* zu Übersetzen. Das Auftreten einer geprüften Exception muss in einem try-catch-Block behandelt werden (oder weiter propagiert werden). Sorgen Sie dafür, dass alle geprüften Exceptions in einem catch-Block aufgefangen werden.

Wenn Sie die Klasse fehlerfrei übersetzen können, sollten Sie probieren, ob nun alle Befehle korrekt funktionieren. Probieren Sie aus, ob sich das Programm bei Eingabe eines leeren Strings sowie eines existenten (z.B. „emma“) und nicht existenten (z.B. „otto“) Schlüssels korrekt verhält.

Tipp: Wenn Sie eine neue geworfene Exception einer Methode in die Javadoc-Dokumentation aufnehmen wollen, dann selektieren Sie den Methodennamen und klicken Sie mit der linken Maustaste auf das Glühbirnensymbol links neben der Zeile. Mit dem blau markierten Vorschlag können Sie die Javadoc Ergänzung einfügen. Vergessen Sie nicht, zu beschreiben, wann die Exception geworfen wird.

Werfen von mehreren geprüften Exceptions

Sehen Sie sich die Methode *deleteKontakt()* in der Klasse *Adressbuch* an. Wenn der angegebene Schlüssel nicht bekannt ist, so wird hier eine *NullPointerException* geworfen. Das ist eigentlich nicht wirklich optimal, da diese ungeprüfte Exception zu einem Programmabbruch führt. In unserer Klientenklasse wird die *NullPointerException* zwar abgefangen, allerdings gibt es keine Gewähr, dass diese Exception auch in einem anderen Kontext bekannt ist und aufgefangen wird.

Besser ist es, auch diesen Fehler durch eine geprüfte Exception zu melden. Dann muss die Methode das Werfen der Exception deklarieren und diese in der Javadoc-Dokumentation der Klasse definieren. Jeder Nutzer der Klasse wird dadurch gezwungen, angemessen auf die Exception zu reagieren.

Schreiben Sie eine zweite Klasse *KeinPassenderKontaktException*, die analog zu der Klasse *UngueltigerSchluesselException* aufgebaut ist. Denken Sie daran, in *toString()* eine verständliche Fehlermeldung auszugeben.

Werfen Sie diesen Fehler anstatt der *NullPointerException* in *deleteKontakt().* Untersuchen Sie die anderen Methoden und überlegen Sie, ob es noch andere Stellen gibt, an denen dieser Fehler geworfen werden sollte. Versuchen Sie die Klasse zu übersetzen und korrigieren Sie eventuelle Fehlermeldungen.

Fangen Sie den neuen Fehler auf geeignete Weise in *AdressbuchTexteingabe.* Testen Sie Ihre Änderungen so wie eben beschrieben. Sind die Ausgaben so, wie Sie es erwarten? Wenn nicht, korrigieren Sie dies.

Hierarchien von Exceptions

Bisher haben wir als ungültige Schlüssel nur Schlüssel betrachtet, die als Wert den leeren String haben. Tatsächlich kann man in unterschiedlichen Situationen verschiedene Spezialfälle von ungültigen Schlüsseln ausmachen:

Leerer String: Das ist immer ein ungültiger Schlüssel

Schlüssel ohne Eintrag im Adressbuch: ungültiger Schlüssel beim Lesen, Ändern oder Löschen eines Kontakts zu diesem Schlüssel (*KeinPassenderKontaktException)*

Schlüssel mit Eintrag im Adressbuch: ungültiger Schlüssel beim Hinzufügen eines neuen Kontakts zu diesem Schlüssel (*DoppelterSchluesselException*).

Erstellen Sie eine Klassenhierarchie für diese verschiedenen Arten von ungültigen Schlüsseln. Als Basisklasse können Sie die Klasse *UngueltigerSchluesselException* nehmen. Leiten Sie davon auf geeignete Weise die Klassen *KeinPassenderKontaktException, DoppelterSchluesselException* ab.

Werfen Sie die neuen Exceptions in den entsprechenden Methoden der Klasse *Adressbuch*.

Bei der Methode *addKontakt()* muss man etwas aufpassen. Hier muss mit Hilfe der Methode *schluesselBekannt(..)* geprüft werden, ob der Name oder die Telefonnummer des Kontakts bereits im Adressbuch als Schlüssel bekannt sind. Die Methode *schluesselBekannt(..)* wirft eine *UngueltigerSchluesselException*, wenn der übergebene Schlüssel ein leerer String ist. Diese Exception darf hier nicht einfach so weiter gereicht werden. Da es reicht, wenn bei einem Kontakt entweder der Name oder die Telefonnummer als Schlüssel eingetragen sind, darf eine der beiden Attribute den leeren String als Wert haben. Die Werte der Attribute des Kontakts werden beim Anlegen eines Kontakts überprüft, es kann also nicht vorkommen, dass beide Schlüssel einen leeren String als Wert haben. Sie müssen daher die *UngueltigerSchluesselException* des Aufrufs von *schluesselBekannt(..)* hier abfangen. Da *DoppelterSchluesselException* eine Subklasse von *UngueltigerSchluesselException* ist, wird auf jeden Fall auch diese Exception mit gefangen. Lesen Sie in den

Vorlesungsfolien nach, wie man im catch-Block eine Fallunterscheidung machen kann um die gefangene *DoppelterSchluesselException* weiter zu leiten.

Bauen Sie in die Klasse *AdressbuchTexteingabe* die entsprechenden catch-Blöcke ein, wo der Compiler sie vermisst. Sie werden feststellen, dass nur eine Stelle angepasst werden muss, da die meisten der Exceptions bereits durch die Superklasse *UngueltigerSchluesselException* mit gefangen werden.

Nutzen Sie die Polymorphie Ihrer Exception-Hierarchie aus, um die catch-Blöcke in *AdressbuchTexteingabe* zu reduzieren.

Wenn alles fehlerfrei ist, dann testen Sie gründlich, ob sich das Programm bei allen Befehlen und Eingaben korrekt verhält. Testen Sie dabei möglichst alle weiter oben genannten Fälle.

Tipp: Prüfen Sie einmal, einen Kontakt zu einem existierenden Namen zu ändern (z.B. chris) und in dem zu verändernden Kontakt einen Namen einzutragen, der ebenfalls im Adressbuch bekannt ist, aber nicht der alte Schlüsselwert ist (z.B. emma). Sehen Sie sich an, was danach im Adressbuch passiert ist. Korrigieren Sie dieses Verhalten in der Methode *updateKontakt(..)* in *Adressbuch*.

Zusatzaufgabe

Bei *aendereEintrag()* muss darauf geachtet werden, dass die textuelle Interaktion nutzerfreundlich funktioniert. Anstatt alle Angaben komplett abzufragen und dann erst den eingegebenen Schlüssel zu prüfen, sollte erst der Schlüssel abgefragt und geprüft werden. Erst wenn ein korrekter und existierender Schlüssel angegeben wurde, sollte der neue Kontakt eingelesen werden. Falls das bei Ihnen noch nicht so läuft, korrigieren Sie das Verhalten.

Beim erfolgreichen Anlegen bzw. Löschen oder Ändern eines Kontakts fehlt noch eine passende Rückmeldung des Programms. Fügen Sie nutzerfreundliche Konsolenausgaben hinzu.

Wer immer noch nicht genug hat

Sie haben es so gewollt: Beim Testen ist mir aufgefallen wie lästig es bei manchen Befehlen ist, dass man erst den Befehl und dann den Schlüssel eingeben muss. Z.B.

```
> hole

Suchschluessel:

otto
```

Ändern Sie die Eingabe von „hole“ und „entferne“ dahingehend, dass man gleich nach dem Befehl den Schlüssel des Kontakts angeben kann, z.B.

```
> hole otto

> entferne otto
```