Programmierung II – Übung 1

Installation von Netbeans und Java

Für das Arbeiten auf Ihrem eigenen Rechner sollten zunächst Netbeans und die Java-Libraries installiert und eingerichtet werden. Die Installationsanleitung finden Sie im Moodle-Projekt unter *Software*.

Projekt Adressbuch-V1T

Starten Sie Netbeans und öffnen Sie mit *File->Open Project* in *Netbeans* das Projekt *Adressbuch_VT1* aus dem Ordner *UBG_1*. In diesem Projekt wird – wie in der Vorlesung – die Klasse *Adressbuch* und die Klasse *Kontakt* als Dienstleister genutzt.

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 noch keinen Gebrauch von den Methoden *getKontakt(..)*, *deleteKontakt(..)* und *updateKontakt(..)*.

Erweitern Sie die Klassen *Befehlswoerter* und *AdressbuchTexteingabe* so, dass der Nutzer auch diese Funktionalität aufrufen kann. Hierzu müssen Sie zunächst den Satz der Befehlswörter um die Worte „hole“, „aendere“ 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()

private void aendereEintrag()
```

Diese Methoden sollen in der while-Schleife von *starten()* aufgerufen werden, wenn der entsprechende Befehl („hole“, „aendere“ bzw. „entferne“) gegeben wurde. Sie können sich hierzu sehr gut an dem Umgang mit dem Befehl „*suche“* mit Hilfe der Methode *sucheEintrag()* orientieren.

Eintrag holen: Die Methode `holeEintrag()` sollte einen Schlüssel einlesen und nur den Kontakt zu dem eingegebenen Schlüssel ausgeben, nicht die Kontakte, die den Schlüssel als Präfix haben.

Eintrag löschen: Die Methode `loescheEintrag()` soll einen Schlüssel einlesen und den Eintrag zu dem Schlüssel aus dem Adressbuch löschen.

Eintrag Ändern: Die Methode `aendereEintrag()` soll einen Schlüssel des zu ändernden *Kontakt*-Objekts und die Informationen des geänderten *Kontakt*-Objekts einlesen und die Änderung dem Adressbuch mitteilen. Überarbeiten Sie in der Klasse *AdressbuchTexteingabe* zunächst die Methode `neuerEintrag()`. Hier werden die einzelnen Attributwerte für ein *Kontakt*-Objekt eingelesen und daraus wird 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, die die einzelnen Attributwerte eines Kontakts einliest, ein neues *Kontakt*-Objekt daraus erstellt und dieses an den Aufrufer zurück liefert. Rufen Sie die Methode in der Methode `neuerEintrag()` auf, um das neu zu erstellende *Kontakt*-Objekt einzulesen. Dabei entfallen viele Anweisungen.

Verwenden Sie die Methode `kontaktEinlesen()` nun auch in der Methode `private void aendereEintrag()`. Nach dem Einlesen des alten Schlüssels und des *Kontakt*-Objekts mit den geänderten Daten soll die Methode *updateKontakt(..)* der Klasse *Adressbuch* aufgerufen werden. Dieser Methode muss der alte Schlüssel des *Kontakt*-Objekts übergeben werden und das neue *Kontakt*-Objekt, das anstelle des alten Objekts gespeichert werden soll.

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 eine entsprechende Fehlermeldung mit.

In unserem Beispiel kann der Fehler des Nutzers sehr wohl korrigiert werden, wenn dieser beim Erstellen eines neuen Kontakts bzw. beim Ändern eines vorhanden Kontakts so lange zu der Eingabe der Kontaktdaten aufgefordert wird, bis ein korrekter Kontakt erzeugt werden konnte. D.h. 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()` der Klasse *AdressbuchTexteingabe* 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 dem Exception-Objekt im Konstruktor mitgegeben wurde.

Probieren Sie aus, ob es funktioniert.

Methodenparameter der Methode *schluesselBekannt()*

In die Methode *getKontakt(..)* der Klasse *Adressbuch* hatten wir in der Vorlesung bereits eine Prüfung des übergebenen Schlüsselwertes eingebaut. Wenn der Schlüssel den Wert *null* oder den leeren String hat, so wird eine *IllegalArgumentException* geworfen. Verschieben Sie diese Prüfung in die Methode *schluesselBekannt(..)* der Klasse *Adressbuch*. Diese Methode beantwortet die Frage, ob der angegebene Schlüssel im Adressbuch existiert. Bevor sie dies tut, soll jetzt geprüft werden, ob es sich bei dem übergebenen Schlüsselwert um einen gültigen Schlüssel handelt.

Da es sinnvoll ist, überall dort, wo ein Schlüssel übergeben wird zu prüfen, ob der Schlüssel überhaupt im Adressbuch existiert, bauen Sie die Methode *schluesselBekannt()* sinnvoll in jede Methode der Klasse *Adressbuch* ein, der ein Schlüssel übergeben wird. So kann sie z.B. in der Methode *deleteKontakt(..)* verwendet werden, um herauszufinden, ob es überhaupt einen Kontakt zu diesem Schlüssel gibt, bevor man versucht den Kontakt zu löschen. Die Methode soll so eingebaut werden, dass sie aufgerufen wird, bevor man versucht, über den Schlüssel auf einen Kontakt zuzugreifen.

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 einem ungültigen Schlüssel (z.B. " ") zu entfernen, aus dem Adressbuch zu lesen, bzw. zu aktualisieren. Wird hier durch die Exception auf den Fehler hingewiesen?

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.

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. So wird z.B. in der Methode *schluesselBekannt(..)* eine *IllegalArgumentException* geworfen, wenn der übergebene Schlüssel entweder den Wert *null* hat oder der Schlüssel ein leerer String bzw. ein String mit nur Leerzeichen ist.

Ein ungültiger Schlüssel (leerer String oder nur Leerzeichen) könnte jedoch vom Nutzer korrigiert werden und sollte 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 oder nur Leerzeichen) 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 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 der Aufrufer nicht davon informiert, dass der Kontakt nicht gelöscht werden konnte.

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, dem Superkonstruktor eine verständliche Fehlermeldung mitzugeben.

Werfen Sie diesen Fehler in *deleteKontakt(),* wenn der angegebene Schlüssel im Adressbuch nicht bekannt ist. 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 oder beim Ändern eines Kontakts, wenn der Name geändert wurde (*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. Sollte dies der Fall sein, so muss eine *DoppelteSchluesselException* geworfen werden. Die Methode *schluesselBekannt(..)* wirft eine *UngueltigerSchluesselException*, wenn der übergebene Schlüssel ein leerer String ist. Diese Exception darf hier nicht einfach so propagiert werden. Da es reicht, wenn bei einem Kontakt entweder der Name oder die Telefonnummer als Schlüssel eingetragen sind, darf eines der beiden Attribute den leeren String als Wert haben. Da die Werte der Attribute des Kontakts beim Anlegen eines Kontakts überprüft werden, kann es 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 nur die gefangene *DoppelterSchluesselException* weiter zu werfen.

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 nach, ob das Adressbuch unverändert geblieben ist und die richtige Fehlermeldung ausgegeben wurde.