Programmierung II –Übung 3

Öffnen Sie mit *File->Open Project* in *Netbeans* das Projekt *Terminverwaltung* aus dem Ordner *UBG_3*. Dieses Projekt enthält den Stand der Adressbuch-Anwendung, wie er nach Durchführung der Übung 2 (ohne Benutzerschnittstelle) sein sollte und ein Paket *terminverwaltung*, wo die Klassen für eine einfache Terminverwaltung zu finden sind.

Terminverwaltung

Zunächst sollten Sie sich aber mit den Klassen des Programms ein wenig vertraut machen. Sehen Sie sich zunächst den Quellcode der Klasse *Termin* an. Diese besitzt diverse Eigenschaften (Attribute): *id* vom Typ *String*, der einen eindeutigen Bezeichner für den Termin enthält, *text* vom Typ *String* für eine Zeichenkette, die Sinn und Zweck des Termins beschreibt, *datum* vom Typ *LocalDate* mit dem das Datum der Verabredung festgelegt werden kann, zwei Attribute *von bis* vom Typ *LocalTime*, mit dem der Start- und Endzeitpunkt definiert werden kann, *besitzer* vom Typ *Kontakt* notiert denjenigen, der den Termin angelegt hat, und *teilnehmer* vom Typ *TreeSet<Kontakt>* enthält alle Teilnehmer des Termins. Die Attribute werden bei der Erzeugung des Objekts über den Konstruktor mit Werten bestückt, wobei die Id des Termins aus Attributwerten zusammengesetzt wird. Sehen Sie sich die Methoden an und versuchen Sie anhand der Kommentare herauszufinden, was die Methoden machen.

Die Klasse *Terminverwaltung* ist etwas komplexer. Hier werden Termine in zwei HashMaps verwaltet. Die *HashMap termineDate* speichert die Termine zu ihrem Datum in einer *ArrayList* ab. D.h. als Schlüssel für die Liste mit den Terminen dient das Datum der Termine. Die *HashMap termineId* enthält alle Termine, abgespeichert unter ihrer Id. Versuchen Sie zu verstehen, was die Methoden der Klasse machen sollen, indem Sie die Kommentare lesen.

Die beiden *Exception*-Klassen modellieren die Fehlersituation für eine Terminüberscheidung (*TerminUeberschneidungException*) und einen ungültigen Termin, d.h. einen Termin bei dem der Endezeitpunkt vor dem Startzeitpunkt liegt (*UngueltigerTerminException*).

Probieren Sie nun die Methoden der Klasse *TesteTerminverwaltung* aus. Dazu können Sie in der main-Methode die einzelnen Methoden aus- und einkommentieren, damit die Tests funktionieren, darf nur immer ein Test auf einmal durchgeführt werden. Sie sollten inzwischen wissen, wie man ein Programm mit einer *main*-Methode in *Netbeans* startet.

Machen Sie sich klar, was in den beiden Methoden getestet wird.

Schreiben Sie eine eigene Testmethode, die testet, ob auch das Ändern eines Termins (*updateTermin(..)*) reibungslos funktioniert. Testen Sie hier zwei Fälle:

1.) Positiv-Test: Es wird nur die Beschreibung des Termins geändert, d.h. das Update müsste funktionieren und der veränderte Termin müsste in der ausgegebenen Liste auftauchen.

2.) Negativ-Test: Setzen Sie bei der Kopie neue Zeiten, so dass sich das Update mit einem existierenden Termin überschneidet. Das müsste eine TerminUeberschneidungsException auslösen. Die Liste der Termine müsste danach unverändert sein.

Tipp: Die Methode *updateTermin(..)* der Klasse *Terminverwaltung* erwartet eine Kopie eines in der Terminverwaltung existierenden Objekts, bei dem bestimmte Attributwerte geändert wurden (durch Aufruf von set-Methoden). Sie können eine Kopie eines *Termin*-Objekts mit Hilfe der Methode *getCopy()* der Klasse *Termin* erzeugen.

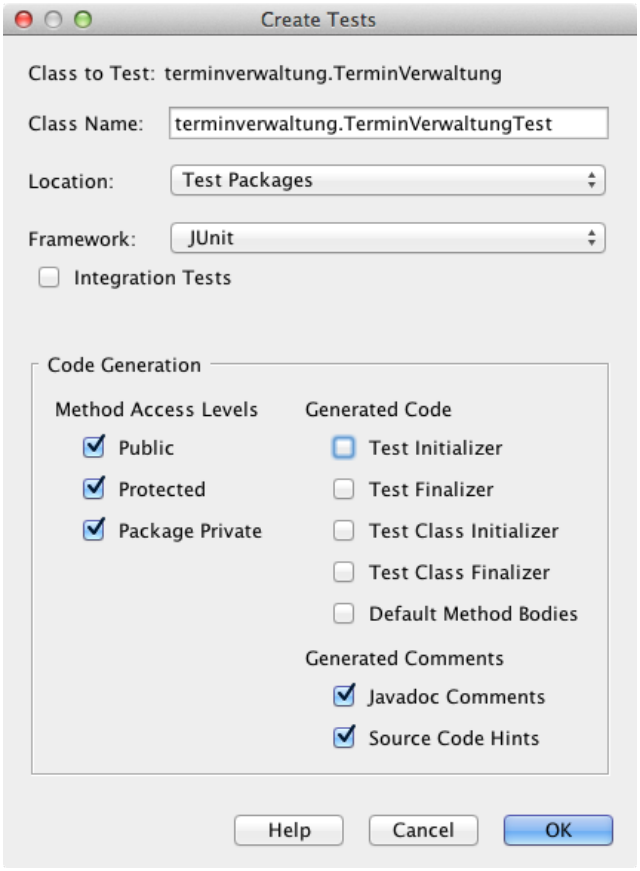
Sie könnten sich jetzt daran machen, die Fehler zu korrigieren. Danach müssten Sie testen, ob Sie das Programm verschlimmbessert haben, d.h. Sie müssten dieselben Tests noch einmal durchführen, um sich abzusichern, dass alles was bereits getestet wurde, immer noch funktioniert. Da die Interpretation der Konsolenausschriften eine aufwändige und fehleranfällige Angelegenheit ist, sollten Sie besser weiter lesen ;-).

Testen mit JUnit

Das erneute Durchführen aller Tests nach einer Änderung im Quellcode nennt man Regressionstests. Diese sind eine sehr mühsame Angelegenheit, wenn man die Ausgaben auf der Konsole stets erneut prüfen und interpretieren muss. Es ist sinnvoller, das Testen weiter zu automatisieren. Das lässt sich ganz wunderbar mit dem Framework *JUnit* erledigen.

Da die Klassen, die das Framework benötigt, nicht in der Java-Standard-Bibliothek zu finden sind, muss dem Projekt zunächst die *JUnit*-Bibliothek hinzugefügt werden. Selektieren Sie dazu links im Projekt-Browser den Ordner *Libraries* des Projekts und wählen Sie im Kontextmenü *Add Library*. Hier werden Ihnen zusätzliche Java-Bibliotheken angeboten. Selektieren Sie *JUnit 4.10* und drücken Sie *Add Library*. Die Bibliothek erscheint nun im Ordner *Libraries*. Wenn Sie den Pfeil aufklappen, dann sehen Sie alle Pakete der neu hinzugefügten Bibliothek.

Sie können für die Klasse *Terminverwaltung* eine *JUnit*-Testklasse von *Netbeans* erzeugen lassen, indem Sie die Klasse auswählen und im Kontextmenü auf *Tools->Create/Update Tests*. Es öffnet sich ein Eingabefenster, in dem Sie folgendes einstellen:

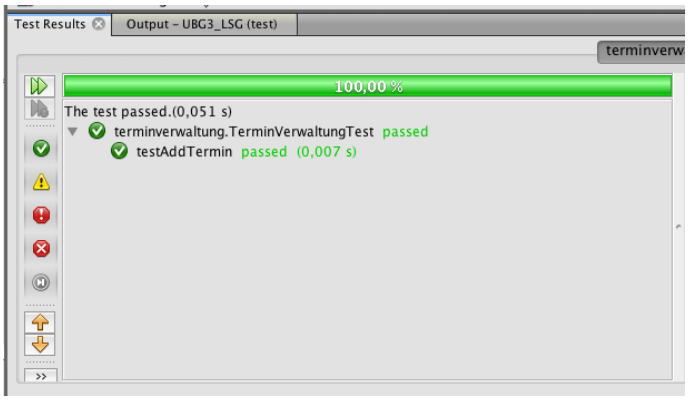


Wenn Sie auf *OK* gehen, erstellt *Netbeans* eine Testklasse *TerminVerwaltungTest* für die Klasse *Terminverwaltung* im Ordner *Test Packages* im Paket *terminverwaltung*.

Öffnen Sie die Testklasse und sehen Sie sich die Methoden genauer an. *Netbeans* hat für alle Methoden der zu testenden Klasse eine Testmethode angelegt, die mit der *@Test* Annotation versehen ist. Das liegt daran, dass Sie im Fenster oben unter *Method Access Level* bei allen Zugriffsmodifikatoren einen Haken gesetzt haben. Hätten Sie nur bei *public* den Haken gesetzt, so gäbe es lediglich für die öffentlichen Methoden der Klasse Testmethoden. Wenn Sie genau hinsehen, erkennen Sie auch, dass bereits alle Assert-Methoden statisch importiert wurden.

Schreiben Sie jetzt die Testmethode *testAddTermin()*. Hier sollten Sie probieren, ob man tatsächlich korrekt einen Termin in den Terminkalender einfügen kann. Sehen Sie im Skript nach welche *Assert*-Anweisungen Sie dafür verwenden können.

Sie können mit *Netbeans* einen Test laufen lassen, indem Sie die zu testende Klasse wählen und im Kontextmenü *Test File* aufrufen. Das Testergebnis wird unter dem Editor angezeigt:



Mit dem grünen Doppelpfeil kann man alle Tests noch einmal durchführen. Wenn Sie alles richtig gemacht haben, dann müsste der Test keinen Fehler melden.

Schreiben Sie nun den Test *testCheckTerminUeberschneidung()* und prüfen Sie, ob die Methode *checkTerminUeberschneidung(..)* der Klasse *TerminVerwaltung* fehlerfrei eine Überschneidung erkennt. Denken Sie daran, Ihre Terminverwaltung und die Termine in der Testmethode neu zu erzeugen. Lassen Sie die Tests noch einmal laufen, wenn Sie vorhin den Fehler nicht korrigiert haben, dann sollte der Test scheitern. Klappen Sie den Test aus, der das gelbe Ausrufezeichen trägt und offenbar fehlgeschlagen ist. Im Fenster wird Ihnen mitgeteilt welcher Wert in der Assertion erwartet wurde und was der tatsächliche Wert war. Sie können selber die Assert-Anweisungen mit Meldungen bestücken, die die Art des Fehlers näher beschreiben, indem Sie der Assert-Anweisung als ersten Parameter eine Zeichenkette mit der gewünschten Meldung mitgeben (die Anführungszeichen nicht vergessen). Probieren Sie es aus!

Schreiben Sie den Test *testGetTermineTag()* der prüft, ob zu einem bestimmten Datum alle Termine korrekt zurück geliefert werden. Falls Sie den Fehler von vorhin noch nicht korrigiert haben, so kommt es hier sogar zu einer *NullPointerException* in der Testmethode selber. Versuchen Sie nun den Fehler zu beheben und lassen Sie die Tests noch einmal laufen.

Testgerüste

Sehen Sie sich Ihre Testmethoden einmal an. Sie werden feststellen, dass in allen Methoden zunächst ein Testszenario mit diversen Objekten aufgebaut werden muss, bevor die entsprechenden Methoden getestet werden können. Das können Sie leichter haben. Eine Methode (meist mit *setUp()* benannt), die mit der Annotation *@Before* markiert ist, kann dazu verwendet werden, solche Testszenarien (Testgerüste) aufzubauen. Diese Methode wird vor jeder Testmethode aufgerufen, so dass stets dasselbe Testszenario zur Verfügung steht.

Legen Sie in der Testklasse Attribute an, die die Objekte aus Ihrem Testszenario aufnehmen können. Dann erstellen Sie in der Methode *setUp()* das Testszenario für Ihre Tests. Denken Sie daran, die Erstellung der Objekte, die nun in der Methode *setUp()* erstellt werden, aus den Testmethoden zu löschen. Probieren Sie aus, ob alles funktioniert. Vergessen Sie die Annotation nicht!

Zusatzaufgabe:

Implementieren Sie auch die restlichen Testmethoden. Denken Sie daran, dass Sie beim Testen der Methode *updateTermin(..)* die Änderungen an einer Kopie des zu verändernden Termins vornehmen (s.o. Tipp).