

Programmierung II – Übung 4

Projekt Taschenrechner

Erstellen Sie in *Netbeans* ein neues Projekt. *UBG4*.

Tipp: Um ein neues Projekt in *Netbeans* anzulegen, wählen Sie im Hauptmenü *File->New Project* und dann im Fenster *New Project* unter *Categories Java* und unter *Projects Java Application* und betätigen Sie *Next*. Im nachfolgenden Fenster können Sie unter *Project Name* den Namen des Projekts angeben. Achten Sie darauf, dass Sie den Pfad unter *Project Location* auf den Ordner setzten, in dem Ihr *Netbeans* Projekt gespeichert werden soll.

Legen sie ein Paket *taschenrechner* an und importieren Sie die Java-Quelldateien in das Paket, indem Sie es aus dem Ordner *Sourcen* in das Paket in *Netbeans* kopieren. Denken Sie daran, die Package-Statements in den drei Dateien anzupassen.

Das Projekt beinhaltet drei Klassen, sehen Sie sich den Quellcode der Klassen an:

Recheneinheit: Dies ist das Herz der Anwendung. Die Klasse bietet Methoden an, die man üblicherweise bei einfachen Taschenrechnern vorfindet. Man kann Ziffern eingeben, addieren, subtrahieren, mit der Methode *gleich()* das Ergebnis ausgeben und mit *clear()* die zuletzt eingegebene Ziffer löschen.

GrafischeSchnittstelle: Die Klasse *Recheneinheit* wird über eine grafische Benutzerschnittstelle bedient. Diese Benutzerschnittstelle wird durch die Klasse *GrafischeSchnittstelle* modelliert. Startet man die main-Methode der Klasse, so wird ein Fenster geöffnet in dem man eine Benutzeroberfläche für einen einfachen Taschenrechner findet. Das Objekt vom Typ *GrafischeSchnittstelle* besitzt ein Attribut *rechner*, dem im Konstruktor ein Objekt vom Typ *Recheneinheit* zugewiesen wurde. Werden Tasten auf der Benutzeroberfläche gedrückt, so werden die zugehörigen Methoden des *Recheneinheit*-Objekts aufgerufen.

RecheneinheitTester: Dies ist die selbst geschriebene Klasse, mit der der Programmierer der Recheneinheit die Klasse getestet hat. Sie wissen bereits aus der Vorlesung, dass die Klasse *Recheneinheit* Fehler enthält, die von der Testklasse nicht aufgedeckt werden.

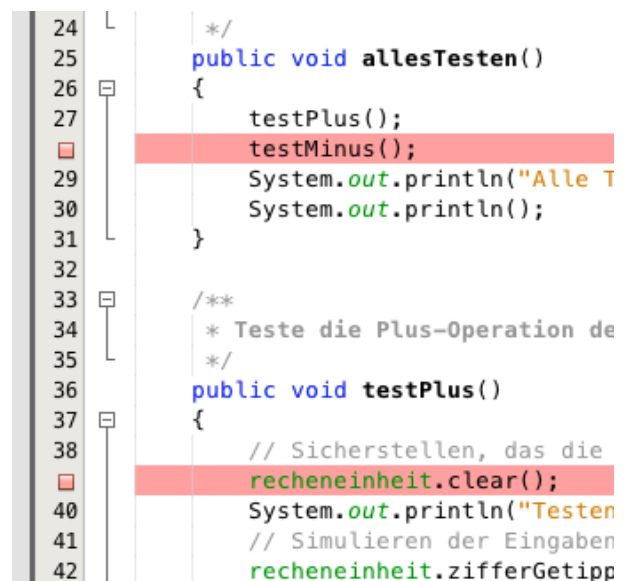
Starten Sie zunächst die Klasse *RecheneinheitTester*, hier wird die Methode *allesTesten()* aufgerufen. Die Ausgabe suggeriert, dass alles funktioniert. Bauen Sie in die Methode *allesTesten()* gleich nach dem Aufruf von *testMinus()* die Methode *testPlus()* ein. Probieren Sie es aus, der ausgegebene Wert sollte Sie stutzig machen. Bauen Sie danach noch einmal *testPlus()* auf. Hier kommt ein anderer Wert heraus.

Debuggen

Sie sollten jetzt den Debugger nutzen, um nachzuvollziehen, was in der Klasse *Recheneinheit* eigentlich genau passiert.

Damit der Debugger in den Ablauf des Programms einsteigt, muss man Haltepunkte (Breakpoints) im Programm setzen. Setzen Sie Methoden-Haltepunkte auf den Methoden *testPlus()* und *testMinus()* in der Klasse *RecheneinheitTester*. Das können Sie machen, indem Sie einen Mausklick in die linke Leiste neben der Zeile mit dem Methodenkopf setzen. Es entsteht dann ein Methoden-Haltepunkt Symbol .

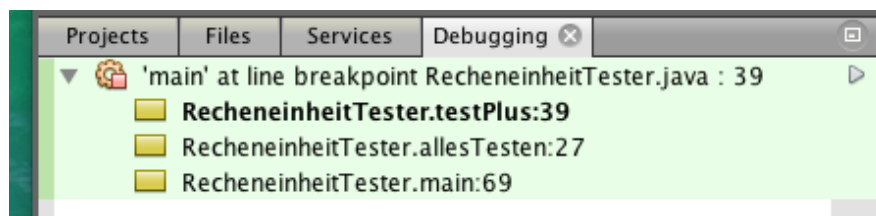
Wir wollen uns mal genauer ansehen, was so alles hinter den Kulissen passiert, wenn man nach einem *testMinus()* ein *testPlus()* aufruft, denn hier scheint etwas schief zu laufen. Setzen Sie daher noch einen Haltepunkt vor dem Aufruf von *testMinus()* in der Methode *allesTesten()*.



```
24  L
25  */
26  public void allesTesten()
27  {
28      testPlus();
29      testMinus();
30      System.out.println("Alle T
31      System.out.println();
32  }
33
34  /**
35   * Teste die Plus-Operation de
36   */
37  public void testPlus()
38  {
39      // Sicherstellen, das die
40      recheneinheit.clear();
41      System.out.println("Testen
42      // Simulieren der Eingaben
43      recheneinheit.zifferGetipp
```

Um den Debugger zu starten selektieren Sie im Projekt Browser die Klasse *RecheneinheitTester.java* und wählen sie im Hauptmenü *Debug File*. Das Programm bleibt am ersten Haltepunkt stehen und der Debugger wartet auf Ihre Angaben.

Links hat sich ein neuer Reiter Debugging geöffnet, in dem der Stapel der Methodenaufrufe angezeigt wird.





Mit einem Doppelklick auf die einzelnen Methodenaufrufe lässt sich der Variablenkontext des jeweiligen Methodenaufrufs einblenden. Dieser wird unter dem Editorfenster im Reiter *Variables* angezeigt. Hier sieht man, dass das aktuelle Objekt vom Typ *RecheneinheitTester* ein Attribut *recheneinheit* vom Typ *Recheneinheit* besitzt, das die drei angezeigten Attribute enthält. Die Werte der Variablen werden ebenfalls angezeigt.

Test Results	Output	Variables	Breakpoints	
		Name		
		Type		
		<Enter new watch>		
		▼ this	RecheneinheitTester	#118
		▼ recheneinheit	Recheneinheit	#121
		◆ anzeigewert	int	0
		◆ letzterOperator	char	' '
		◆ linkerOperand	int	0


Oben in der Werkzeugleiste von *Netbeans* werden die Buttons für die Bedienung des Debuggers angezeigt.




Da wir hier anfänglich vor der Abarbeitung der *testMinus()* Methode stehen, vor der wir sowieso einen Haltepunkt gesetzt haben, können wir mit dem Fortsetzen-Button 

diese Methode anspringen. Alternativ kann man hier *Schritt hinein* (*Step into*) wählen (siehe unten). 

Sie können jetzt entscheiden, ob Sie den Ablauf der Methode *clear()* Schritt für Schritt verfolgen wollen, oder ob Sie die Methode überspringen. Die Methode *clear()* können Sie am Anfang der Berechnungen getrost überspringen. Wählen Sie dazu den Button

Schritt über (*Step over*) . Der Debugger stoppt jetzt vor der nächsten Anweisung, überspringen Sie auch diese, da es sich um eine Ausgabeanweisung handelt.

In die nächste Anweisung kann man mal hinein gehen und sich den Ablauf genauer

ansehen. Wählen Sie *Schritt hinein* (*Step into*) , um in die Methode *zifferGetippt(..)* hineinzusteigen. Nun befinden Sie sich in der Methode *zifferGetippt(..)* auf der ersten (und einzigen) Anweisung der Methode. Die Anweisung ist noch nicht ausgeführt worden. Inspizieren Sie die Angaben zu den Variablen unter dem Editor. Um die Werte der Attribute der aktuellen Instanz der Klasse *Recheneinheit* zu sehen, müssen Sie die Zeilen unter *this* ausklappen. Hier können Sie sehen, mit welchen Werten die Attribute des Objekts belegt sind (Zustand) und welche Werte die lokalen Variablen der Methode haben (hier gibt es nur den einen Methodenparameter *ziffer*).

Wählen Sie *Schritt über* um die Methode auszuführen und sehen Sie sich an, wie sich dadurch die Belegung der Attribute verändert. Die veränderten Werte werden fett markiert angezeigt.

Wählen Sie *Schritt über*, bis Sie auf der nächsten Anweisung der Methode *testMinus()* sind. Steigen Sie auch in die Anweisung *recheneinheit.minus()* hinein und steigen Sie

dort auch in die Methode *letztenOperatorAnwenden()* ab. Beobachten Sie genau, welche Anweisungen ausgeführt werden und wie sich bei jedem Schritt die Instanzvariablen ändern.

Kehren Sie mit *Schritt über* (u.U. mehrmals) zurück in die Methode *testMinus()*. Sie sollten sich jetzt auf der Anweisung *zifferGetippt(4)* befinden. Steigen Sie auch hier ab und analysieren Sie, was die Recheneinheit tut.

Die nächste Anweisung in *testMinus()* ist sehr spannend. Mit dem Aufruf der Methode *gleich()* wird die eingegebene Operation ausgeführt. Steigen Sie in *gleich()* hinab und beobachten Sie genau, was passiert. Sehen Sie sich hier vor allen Dingen den Endzustand der Attribute an, nachdem die letzte Anweisung von *gleich()* ausgeführt wurde.

Betätigen Sie *Schritt über*, bis Ihnen das Ergebnis der Operation in der Konsole unten (Um die Ausgabe zu sehen, müssen Sie auf den Reiter *Output* wechseln) mitgeteilt wird und Sie wieder in der Methode *allesTesten()* landen. Hier steht der Debugger jetzt vor dem Aufruf von *testPlus()* nach Ausführung von *testMinus()*.

Steigen Sie nun in die Methode *testPlus()* ein und analysieren Sie, was in der Klasse Recheneinheit passiert. Steigen Sie als erstes in die Methode *clear()* hinab. Achten Sie sorgfältig auf die Belegung der Attribute. Beim Aufruf von *plus()* sollten Sie in die Methode *letztenOperatorAnwenden()* einsteigen um herauszufinden was hier schief läuft.

Tipp: Um die Haltepunkte in Ihrem Quellcode wieder zu entfernen, reicht es aus, mit der Maus auf den Haltepunkt zu klicken.

Testklasse Schreiben

Ganz klar ist: Dieses Programm läuft falsch und die Testklasse *RecheneinheitTester* taugt auch nicht viel, denn sie hat den Fehler nicht aufgespürt.

Das können Sie besser. Schreiben Sie eine *JUnit*-Testklasse, die die Klasse Recheneinheit einem vernünftigen Test unterzieht. Wenn Sie nicht mehr genau wissen, wie man ein Gerüst für eine Testklasse von *Netbeans* generieren lassen kann, dann sehen Sie in der letzten Übung nach.

Wenn Sie mit der Testklasse soweit sind und der Test tatsächlich den eben beobachteten Fehler aufspüren kann, dann versuchen Sie den Fehler in *Recheneinheit* zu korrigieren. Ob Ihre Korrekturen den Fehler behoben haben, können Sie anhand Ihrer Testklasse nun leicht feststellen.

Test-First

Diese vielbeachtete Technik des *Extreme Programming* dreht diesen Prozess um. Man entwickelt zunächst einen *JUnit*-Test für eine Methode, die man programmieren möchte

und programmiert die Methode erst, wenn der Test bereits geschrieben ist. Diese *Test-First* Technik sollen Sie im Folgenden ausprobieren.

Die besseren Taschenrechner bieten die Möglichkeit, einen Anzeigewert zwischen zu speichern, um ihn zu einem späteren Zeitpunkt wieder in die Rechnung mit einzubeziehen. Das Speichern des aktuellen Anzeigewertes soll mit Hilfe der Methode

```
public void speicherEingabe()
```

erfolgen können. Das Ausgeben des gespeicherten Wertes in die aktuelle Anzeige soll mit Hilfe der Methode

```
public void speicherAusgabe()
```

erfolgen. Erstellen Sie zunächst in der Klasse *Recheneinheit* die entsprechenden Methoden, lassen Sie die Methodenrumpfe leer.

Schreiben Sie jetzt geeignete Testmethoden in der Klasse *RecheneinheitTest*, mit denen Sie verschiedene Testfälle abdecken können. Denken Sie daran, verschiedene Kombinationen zu testen, z.B. Wert berechnen, speichern, *clear()* aufrufen, gespeicherten Wert einsetzen, oder gespeicherten Wert als Operanden einsetzen usw. Lassen Sie Ihre Tests durchlaufen. Sie werden feststellen, dass die neuen Tests noch nicht erfolgreich ablaufen.

Schreiben Sie nun die Methodenrumpfe für die neuen Methoden in *Recheneinheit*. Lassen Sie die Tests laufen und prüfen Sie, ob Ihre Methoden korrekt arbeiten.

Fügen Sie eine Methode zu *Recheneinheit* hinzu, mit der sich der Speicher wieder zurücksetzen lässt. Fügen Sie der Testklasse eine geeignete Testmethode hinzu und testen Sie, ob nach wie vor alles funktioniert.

Zusatzaufgabe

Eigentlich funktioniert das mit dem Speicher in einem Taschenrechner anders. Hier kann mit der Taste *M+* der Anzeigewert zu dem Speicherwert hinzuaddiert und mit *M-* vom Speicherwert abgezogen werden. Mit *Mr* kann der Speicherinhalt abgerufen und mit *Mc* gelöscht werden. Erweitern Sie die Klasse *Recheneinheit* dahingehend, dass diese Funktionalität unterstützt wird. Schreiben Sie vorher Testmethoden, die die neue Funktionalität entsprechend testen.

Wer immer noch nicht genug hat

Wenn Sie bei der Korrektur von *Recheneinheit* nur die Fehler korrigiert haben, dann bleiben die Methoden noch stellenweise unverständlich. Vielleicht haben Sie eine Idee, wie man die Klasse und die Methoden nachvollziehbarer gestalten kann. Modellieren Sie die Klasse dementsprechend um.