Grundlagen der Informatik 1
WS 2008/09

Prof. Mühlhäuser, Dr. Rößling, Melanie Hartmann, Daniel Schreiber
http://proffs.tk.informatik.tu-darmstadt.de/gdi1

Übung 13 Version: 1.0 02.02.2009

---

1 Mini Quiz

1. Generics
   - Gegeben eine Klasse Shape mit Unterklassen Circle und Rectangle. Welche Typen kann man in einer List<? **extends** Shape> speichern?

     ☐ Shape ☐ Object ☐ Rectangle ☐ String
   - Welche Werte sind für den Platzhalter im Ausdruck Vector<Platzhalter> möglich?

     ☐ ? ☐ A **extends** B ☐ A **implements** B ☐ A
2. Assertions

   ☐ Assertions ermöglichen das Prüfen von Vorbedingungen.

   ☐ Die Fehler die von Assertions ausgelöst werden, sind vom Typ `Exception` und nicht vom Typ `Error`.
3. JUnit / Testing

   ☐ JUnit eignet sich zum Debuggen von komplexen Java-Programmen.

   ☐ Besteht ein System alle Unit-, Integrations- und Systemtests, so kann man davon ausgehen, dass es fehlerfrei ist.

   ☐ Kontrollfluss-orientierte Testverfahren erfordern es, den Programmcode zu kennen.

2 Fragen

1. Nennen Sie einige Vor- und Nachteile der statischen Typprüfung gegenüber dynamischer Typprüfung.
2. Was sind die Hauptunterschiede zwischen White-Box Tests und Black-Box Tests?
3. In welcher Phase der Software-Entwicklung sollte man mit dem Testen beginnen? Wann kann man mit dem Testen aufhören und woher weiß man, dass man keine weiteren Tests mehr benötigt?
4. Wann sollte man Assertions verwenden? Wann sollte man stattdessen lieber Exceptions benutzen?
5. Welchen Vorteil bieten Generics? Welches Problem wird dadurch gelöst?

3 Verifikation und Validierung

Zunächst: Ein kleines Beispiel für kritische Fehler in Software...

Im Bereich der Medizin gab es in den Jahren 1985 bis 1987 auf Grund von Softwarefehlern viele Todesfälle. Mehrere Patienten starben, nachdem sie zur Krebsbehandlung mit Therac-25 bestrahlt wurden. Wegen einer zu hohen Strahlendosis wurden ihre Zellen nachhaltig geschädigt.... (Das medizinische Gerät Therac-25 war ein linearer Teilchenbeschleuniger zur Strahlentherapie für krebskranke Patienten.)

Wir betrachten nun nochmals die beiden Begriffe aus der Vorlesung: Verifikation und Validierung. Schauen Sie sich bitte folgenden Text an:

Während die _____ den Output einer Entwicklungsphase auf die Korrektheit mit der vorherigen Phase untersucht, wird die _____ benutzt, um das Gesamtsystem mit den Kundenanforderungen zu vergleichen.
Die zentrale Tätigkeit bei der _____ ist das Testen. Das Gesamtsystem wird bei Ende des Prozesses getestet, ob es der Spezifikation entspricht oder nicht. Die zentrale Tätigkeit bei der _____ ist der Beweis mit der formalen _____.

Setzen Sie die passenden Begriffe (Verifikation / Validierung) korrekt ein.

4 Fehler

Starten Sie bitte Eclipse, legen Sie eine einfache Klasse an und rufen Sie die unten angegebene Funktion auf (wird im Portal bereitgestellt als eclipse.java). Welches Ergebnis erhalten Sie? Wenn Sie keinen Computer dabei haben, fragen Sie Ihren Tutor nach der Lösung.

Was für ein Ergebnis hätten Sie erwartet? Was für ein Bug hat sich hier eingeschlichen und wie könnte man diesen beheben?

```
1 public void hmmmm()
2 {
3         double test = 0;
4
5         test += 277.04;
6         test += 222.67;
7
8         System.out.println(test);
9 }
```

5 Generics

1. Gegeben sei eine Klassenhierarchie für Nahrungsmittel mit der Basisklasse *Nahrungsmittel* und ihren Unterklassen *Obst, Fleisch* und *Gemuese*. Die Klasse *Obst* hat zusätzlich noch die Unterklasse *Apfel*.

   Die Methode

double getNV(LinkedList<Nahrungsmittel>)

berechnet den Gesamtkaloriengehalt der ausgewählten Nahrungsmittel.

Geben Sie für jedes Element der folgenden Tabelle an, ob die entsprechende Operation *legal* (T) ist oder nicht (F). So soll in der ersten freien Zelle der ersten Zeile ein T stehen, wenn man die Methode *getNV* mit einem Parameter vom Typ LinkedList auch mit einer Instanz von LinkedList<Nahrungsmittel> aufrufen darf.

Statischer Typ v in LinkedList<v>→ **Methodendeklaration ↓**	<Nahrungsmittel>	<Obst>	<Apfel>
double getNV(LinkedList x)			
double getNV(LinkedList <Obst> x)			
double getNV(LinkedList <?> x)			
double getNV(LinkedList <? **extends** Obst> x)			
double getNV(LinkedList <? **super** Obst> x)			

2. Gegeben sein nun der folgende Code:

```
1 public void x(ListDef v) {
2   --??-- x = v.get(0);
3   v.add(new Apfel("Boskoop", 43));
4 }
```

In der folgenden Tabelle finden Sie in jeder Zeile wieder eine konkrete Deklaration des Parametertyps *ListDef*.

In der zweiten Spalte tragen Sie bitte ein, welcher Typ für *–??–* in Codezeile 2 eingesetzt werden sollte. Dabei sollten Sie einen möglichst präzisen Typ angeben und daher `Object` nur dann angeben, wenn es keinen konkreteren Typ gibt. Gehen Sie dabei davon aus, dass die Liste existiert und es ein Element an der Position 0 gibt.

In der letzten Spalte geben Sie bitte an, ob die Einfügeoperation in Codezeile 3 jeweils zulässig ist. Verwenden Sie auch hierfür T bzw. F.

Konkreter Typ für ListDef	Typ von --??-- in Zeile 2	v.add(new Apfel(...))
void x(LinkedList v)		
void x(LinkedList <Apfel> v)		
void x(LinkedList <?> v)		
void x(LinkedList <? **extends** Apfel> v)		
void x(LinkedList <? **super** Apfel> v)		

3. Im Rahmen des Sonderangebots "2 Essen zum Preis von einem" werden zwei Mahlzeiten zusammengefügt. Hierfür gibt es die Methode *createMenu()*, die aus den zwei übergebenen *LinkedList*-Objekten eine *neue* LinkedList des *am besten passenden Typs* machen soll:
   - Das Zusammenlegen einer LinkedList<Gemuese> mit einer LinkedList<Fleisch> führt zu einer LinkedList<Nahrungsmittel>.
   - Das Zusammenlegen von *Äpfeln* mit anderem *Obst* führt zu einer LinkedList<Obst>.
   - Werden nur Listen gleichen Typs zusammengelegt, soll am Ende eine Liste des gleichen Typs entstehen.

   Lösen Sie das Problem, indem Sie für die unten angegebene (korrekte) generische Implementierung die passenden Typen *T, U, V* spezifizieren. Begründen Sie in zwei bis drei Sätzen Ihre Typwahl.

```
1 public<T..., U..., V...> LinkedList<V> createMenu
2                 (LinkedList<T> menuA, LinkedList<U> menuB){
3     LinkedList<V> totalMenu = new LinkedList<V>();
4     totalMenu.addAll(menuA);
5     totalMenu.addAll(menuB);
6     return totalMenu;
7 }
```

   Korrekte Deklaration des generischen Typs **T**:

   Korrekte Deklaration des generischen Typs **U**:

   Korrekte Deklaration des generischen Typs **V**: