## Aufgabe 5-2

(Übungsblatt 5)

Sind folgende Java-Anweisungen fehlerfrei zu compilieren und auszuführen? Begründen Sie Ihre Antwort.

a. `int a;a = 5;`
b. `int b = 5;`
c. `int c,c = 5;`
d. `float d = 5,5;`
e. `float e = 5;`
f. `double f1 = 5d; float f2 = f1;`
g. `double g1 = 5e; float g2 = g1;`
h. `double h1 = 5f; float h2 = h1;`
i. `long i = 0.0;`
j. `float j = 0.0;`
k. `float k1 = 5f; double k2 = k1;`
l. `double l = 0.0;`
m. `long m = (short)Integer.MAX_VALUE;`
n. `short n = (long)0;`

### Lösungsvorschlag

zu den Teilaufgaben:

a. Fehlerfrei: Umgestellt ist das gleichbedeutend mit dem folgenden Code-Fragment, welches offensichtlich richtig ist, weil sowohl `a` als auch `5` vom Typ `int` sind:
```
int a;
a = 5;
```
b. Fehlerfrei: sowohl `b` als auch `5` sind vom Typ `int`.
c. Nicht fehlerfrei: Umgestellt ist das gleichbedeutend mit dem folgenden Code-Fragment, welches offensichtlich syntaktisch falsch ist, weil zweimal eine Variable mit Namen `c` deklariert wird:
```
int c;
int c = 5;
```
d. Nicht fehlerfrei: `5,5` ist kein numerischer Wert und kann folglich nicht zugewiesen werden -- `d` hat den Typ `float`, einen Kommazahlentyp. Kommazahlen in Java haben immer einen Dezimal*punkt*.
e. Fehlerfrei: `5` ist zwar vom Typ `int`, aber die Zuweisung eines Ganzzahl-Werts zu einer Kommazahl-Variable ist logischerweise möglich. Es kommt dann zu einer impliziten Konvertierung (Type-Cast); man sagt umgangssprachlich, die Typen `float` und `int` seien in diesem Zusammenhang *zuweisungskompatibel* (gilt aber nicht umgekehrt!).
f. Nicht fehlerfrei. Die zweite Anweisung ist das Problem; in der ersten ist `5d` die Zahl `5` als Wert vom Typ `double` völlig korrekt. Dann aber wird ein `double`-Wert in Form der Variablen `f1` der `float`-Variablen `f2` zugewiesen. Weil `double` doppelt so viele Stellen hat wie `float`, verbietet der Compiler diese Zuweisung. (Mit einem expliziten Type-Cast wäre es möglich, sie zu erzwingen.)
g. Nicht fehlerfrei: `5e` ist kein Wert; die nachgestellten Kürzel gibt es nur für die Typen `float` (`f`), `double` (`d`) und `long` (`L`).
h. Nicht fehlerfrei: analog zu Teilaufgabe (f); dass in der ersten Anweisung rechts ein `float`-Wert steht, ändert nichts daran, dass `h1` vom Typ `double` ist. Es findet eine transparente Typumwandlung statt, genau wie bei Teilaufgabe (e).
i. Nicht fehlerfrei: analog zu Teilaufgabe (f); `0.0` hat den Typ `double`. Kommazahlen-Werte können nicht Ganzahl-Variablen zugewiesen werden, außer, man macht einen expliziten Type-Cast. Teilaufgabe (e) demonstriert den umgekehrten Fall (allerdings mit `float` und `int`).
j. Nicht fehlerfrei: analog zu Teilaufgabe (f); `0.0` hat den Typ `double`, benötigt wird aber `float`.
k. Fehlerfrei: `float` hat weniger Stellen als `double`, daher ist die Zuweisung kein Problem. Die zusätzlichen Stellen in `double` werden quasi einfach mit Nullen aufgefüllt. Vergleiche Teilaufgabe (f).
l. Fehlerfrei: `l` ist vom Typ `double`, `0.0` ebenfalls.
m. Fehlerfrei: `Integer.MAX_VALUE` hat den Wert `2147483647` (natürlich vom Typ `int`). Mit einem expliziten Type-Cast wie hier gezeigt ist es möglich, diesen Wert in den Typ `short` „hineinzupressen," obwohl der viel weniger Stellen hat. Alles, was nicht reinpasst, fällt dann einfach weg. Und den dann rechts resultierenden Wert vom Typ `short` kann man natürlich einer `long`-Variablen zuweisen; die zusätzlichen Stellen in `long` werden dann einfach mit Nullen gefüllt. (`m` hat übrigens hinterher den Wert `-1` wegen der internen Darstellungsform negativer Zahlen. Wie man sieht, warnt der Compiler nicht ohne Grund vor „possible loss of precision.")
n. Nicht fehlerfrei: Rechts steht ein Wert vom Typ `int`, der dann noch explizit in `long` gecastet wird (also noch mehr Stellen bekommt). Der Typ `short` hat aber weniger Stellen als jeder der beiden, so dass eine Zuweisung nur mit explizitem Type-Cast zu `short` möglich wäre.

**Anmerkung.** Bei Problemen dieser Art ist das „Code Pad" in BlueJ äußerst hilfreich. Siehe Unterlagen.

*Arne Johannessen, 2. Dezember 2007*

**Lösungsvorschlag zu Aufgabe 5-3 (a)**

```
 1: /* $Id: Position.java $
 2:  */
 3:
 4:
 5: public class Position {
 6:
 7:     protected double y;
 8:
 9:     protected double x;
10:
11:     public Position (double y, double x) {
12:         this.y = y;
13:         this.x = x;
14:     }
15:
16: }
```

**Lösungsvorschlag zu Aufgabe 5-3 (b)**

```
 1: /* $Id: Position.java $
 2:  */
 3:
 4:
 5: public class Position {
 6:
 7:     protected double y;
 8:
 9:     protected double x;
10:
11:     public Position (double y, double x) {
12:         this.y = y;
13:         this.x = x;
14:     }
15:
16: }
```

```
 1: /* $Id: GeographicPosition.java $
 2:  */
 3:
 4:
 5: public class GeographicPosition extends Position {
 6:
 7:     public GeographicPosition (double lambda, double phi) {
 8:         super(lambda, phi);
 9:     }
10:
11:     public GeographicPosition (double lambdaDegrees, double lambdaMinutes, double phiDegrees,
    double phiMinutes) {
12:         super(Double.NaN, Double.NaN);
13:         this.setLambda(lambdaDegrees, lambdaMinutes);
14:         this.setPhi(phiDegrees, phiMinutes);
15:     }
16:
17:     public void setLambda (double degrees, double minutes) {
18:         super.y = degrees + minutes / 60.0d;
19:     }
20:
21:     public void setPhi (double degrees, double minutes) {
22:         super.x = degrees + minutes / 60.0d;
23:     }
24:
25: }
```

**Lösungsvorschlag zu Aufgabe 5-3 (c)**

```
 1: /* $Id: Position.java $
 2:  */
 3:
 4:
 5: public class Position {
 6:
 7:     protected double y;
 8:
 9:     protected double x;
10:
11:     public Position (double y, double x) {
12:         this.y = y;
13:         this.x = x;
14:     }
15:
16:     public double distanceFrom (Position other) {
17:         double dY = other.y - this.y;
18:         double dX = other.x - this.x;
19:         return Math.sqrt(dY*dY + dX*dX);  // Pythagoras
20:     }
21:
22: }
```

```
 1: /* $Id: GeographicPosition.java $
 2:  */
 3:
 4:
 5: public class GeographicPosition extends Position {
 6:
 7:     public GeographicPosition (double lambda, double phi) {
 8:         super(lambda, phi);
 9:     }
10:
11:     public GeographicPosition (double lambdaDegrees, double lambdaMinutes, double phiDegrees,
    double phiMinutes) {
12:         super(Double.NaN, Double.NaN);
13:         this.setLambda(lambdaDegrees, lambdaMinutes);
14:         this.setPhi(phiDegrees, phiMinutes);
15:     }
16:
17:     public void setLambda (double degrees, double minutes) {
18:         super.y = degrees + minutes / 60.0d;
19:     }
20:
21:     public void setPhi (double degrees, double minutes) {
22:         super.x = degrees + minutes / 60.0d;
23:     }
24:
25: }
```

## Aufgabe 5-4

(Übungsblatt 5)

Betrachten Sie Ihre Klassenstruktur aus Aufgabe 5-3 und das folgende Code-Fragment.

```
GeographicPosition p1, p2;
…  // Konstruktion der Objekte
System.out.println(p1.distanceFrom(p2));
```

a. Ist dieses Code-Fragment syntaktisch korrekt? Begründen Sie Ihre Antwort.
b. Ist dieses Code-Fragment semantisch korrekt? Begründen Sie Ihre Antwort.
c. Was folgern Sie aus dieser Situation in Bezug auf das Prinzip der Vererbung in der objektorientierten Programmierung?

### Lösungsvorschlag

zu den Teilaufgaben:

a. Das Code-Fragment ist syntaktisch korrekt. Nachgewiesen werden kann dies am Einfachsten durch Compilieren; sofern diese Anweisungen dazu in eine Methode geschrieben werden (wo sie nämlich hingehören, wie alle Anweisungen) und die Klassen `GeographicPosition` und `Position` beide zur Verfügung stehen, treten keine Fehler zur Compile-Zeit auf.

   Ein formaler Nachweis wäre mit Hilfe der Java Language Specification möglich.

   Für uns genügt sicherlich eine grobe Beschreibung der einzelnen Anweisungen. In der ersten Zeile werden zwei (lokale) Variablen vom Typ `GeographicPosition` deklariert. Die Variablen werden nicht gleichzeitig initialisiert; eine Zuweisung entsprechender Objekte erfolgt im zweiten (nicht dargestellten) Schritt. Die letzte Zeile ruft die Objektmethode `distanceFrom(Position)` auf und gibt den Rückgabewert auf der Standardausgabe aus.

   Dieser Methodenaufruf ist erlaubt, weil `Position` eine Superklasse von `GeographicPosition` ist (`class GeographicPosition extends Position`). Anschaulich kann man dieses Verhältnis auch anders formulieren: *Jede geographische Position ist gleichzeitig auch eine (allgemeine) Position.*

b. Wie die Entfernung zwischen zwei *nicht näher spezifizierten* Koordinatenpaaren ermittelt wird, hängt immer von der Art der Koordinaten ab. Kartesische Koordinaten werden zum Beispiel anders behandelt als Polarkoordinaten. Folglich ist es nicht möglich, eine Methode zur Entfernungsberechnung aus nichts weiter als zwei beliebigen Koordinatenpaaren *allgemeingültig* anzugeben.

   Im Lösungsvorschlag zu Aufgabe 5-3 (c) wurde von kartesischen Koordinaten ausgegangen und folglich der Satz des Pythagoras verwendet. Dies ist für geographische Koordinaten offensichtlich unzulässig, weil es zu einem falschen Ergebnis führt. (Beispiel: $\phi_1 = 60°$, $\lambda_1 = 0°$, $\phi_2 = 60°$, $\lambda_2 = 10°$ ergibt eine mit Pythagoras errechnete Distanz von 10° [≈ 1100 km auf der Erde], aber die tatsächliche Entfernung beträgt auf der Erde nur etwa 550 km [5°].)

   Folglich ist dieser Code-Abschnitt *nicht* semantisch korrekt.

c. Offensichtlich ist das objektorientierte Prinzip der Erweiterung von Klassenstrukturen durch Vererbung nicht immer problemlos.

   Das Problem war in dieser Klassenstruktur das Hinzufügen der Methode für die Entfernungsberechnung zur allgemeinen Oberklasse `Position`. Instanzen der Unterklassen erben solche neuen Methoden automatisch – selbst dann, wenn sie im Kontext der Unterklasse gar nicht sinnvoll sind.

   Solche Probleme lassen sich umgehen, indem man Klassenstrukturen sorgfältig plant. In Fällen wie dem vorliegenden würde man für jede Art von Koordinaten eine eigene Unterklasse von `Position` erstellen und `Position` selbst dann so deklarieren, dass sie nicht instanziiert werden kann (also nur Objekte der verschiedenen Unterklassen konstruiert werden können). (Dazu setzt man entweder das Zugriffsrecht aller Konstruktoren auf `private` oder man deklariert die Klasse als abstrakt (`abstract class`).)

   (Eine moderne Lösung dieser Problematik stellt das Delegation-Pattern dar; dabei gehen allerdings die Vorteile des Polymorphismus teilweise verloren.)

*Arne Johannessen, 4. Dezember 2007*

**Lösungsvorschlag zu Aufgabe 5-5 (a)**

```
 1: /* $Id: Karte.java $
 2:  */
 3:
 4:
 5: public class Karte {
 6:
 7:     protected int lfdNummer;
 8:
 9:     protected String titel;
10:
11:     protected Object modul;
12:
13:     protected int breite;
14:
15:     protected int hoehe;
16:
17:     protected String zustand;
18:
19:     protected java.util.Date ausgabe;
20:
21:     protected String inhalt;
22:
23:     protected double nettoPreis;
24:
25: }
```

**Lösungsvorschlag zu Aufgabe 5-5 (b)**

```
 1: /* $Id: Abteilung.java $
 2:  */
 3:
 4:
 5: public class Abteilung {
 6:
 7:     protected String name;
 8:
 9:     protected int budget;
10:
11: }
```

```
 1: /* $Id: Mitarbeiter.java $
 2:  */
 3:
 4:
 5: public class Mitarbeiter {
 6:
 7:     protected String name;
 8:
 9:     protected String vorname;
10:
11:     protected int stellenNummer;
12:
13:     protected java.util.Date geburtsdatum;
14:
15:
16:     protected Mitarbeiter vorgesetzter;
17:
18:     protected Abteilung[] abteilungen;
19:
20: }
```

**Lösungsvorschlag zu Aufgabe 5-5 (c)**

```
 1: /* $Id: Start.java Exp $
 2:  */
 3:
 4:
 5: import java.util.GregorianCalendar;
 6:
 7:
 8: public class Start {
 9:
10:     public static void main (String[] args) {
11:
12:
13:         // erst die Karten:
14:
15:         Karte karte1 = new Karte();
16:         Karte karte2 = new Karte();
17:
18:         karte1.lfdNummer = 2007054;
19:         karte1.titel = "More Soer, Projected 4D Lines Block 62/3";
20:         karte1.breite = 687;
21:         karte1.hoehe = 581;
22:         karte1.zustand = "Vertrieb";
23:         karte1.ausgabe = new GregorianCalendar(2007, 8, 26).getTime();
24:         karte1.inhalt = "Panorama";
25:         karte1.nettoPreis = 152.99;
26:
27:         karte2.lfdNummer = 2006141;
28:         karte2.titel = "Nguma, Gabun Offshore";
29:         karte2.breite = 1023;
30:         karte2.hoehe = 795;
31:         karte2.zustand = "nicht mehr im Vertrieb";
32:         karte2.ausgabe = new GregorianCalendar(2007, 2, 13).getTime();
33:         karte2.inhalt = "Touristik";
34:         karte2.nettoPreis = 220.00;
35:
36:
```

(fortgesetzt)

```
37:
38:
39:         // jetzt die Abteilungen und Mitarbeiter:
40:
41:         Abteilung herstellung = new Abteilung();
42:         Abteilung vertrieb = new Abteilung();
43:
44:         Mitarbeiter oberboss = new Mitarbeiter();
45:         Mitarbeiter herstellungsleiter = new Mitarbeiter();  // Abteilungsleiter herstellung
46:         Mitarbeiter vertriebsleiter = new Mitarbeiter();  // Abteilungsleiter vertrieb
47:         Mitarbeiter kartendesigner = new Mitarbeiter();  // Mitarbeiter in herstellung
48:         Mitarbeiter katalogdesigner = new Mitarbeiter();  // Mitarb. in beiden Abteilungen
49:
50:
51:
52:         // Relationen setzen:
53:
54:         oberboss.vorgesetzter = null;
55:         oberboss.abteilungen = new Abteilung[0];  // steht ueber allem
56:
57:         herstellungsleiter.vorgesetzter = oberboss;
58:         herstellungsleiter.abteilungen = new Abteilung[] {herstellung};
59:
60:         vertriebsleiter.vorgesetzter = oberboss;
61:         vertriebsleiter.abteilungen = new Abteilung[] {vertrieb};
62:
63:         kartendesigner.vorgesetzter = herstellungsleiter;
64:         kartendesigner.abteilungen = new Abteilung[] {herstellung};
65:
66:         katalogdesigner.vorgesetzter = vertriebsleiter;  // nur ein Vorgesetzter geben
67:         katalogdesigner.abteilungen = new Abteilung[] {herstellung, vertrieb};
68:
69:
70:
71:         // Stammdaten setzen:
72:
73:         herstellung.name = "KDH";
74:         herstellung.budget = 80000;
75:
76:         vertrieb.name = "Vertrieb";
77:         vertrieb.budget = 25000;
78:
79:         oberboss.name = "Wups";
80:         oberboss.vorname = "Willi";
81:         oberboss.stellenNummer = 10001;
82:         oberboss.geburtsdatum = new GregorianCalendar(1968, 7, 25).getTime();
83:
84:         herstellungsleiter.name = "Bockwurst";
85:         herstellungsleiter.vorname = "Barny";
86:         herstellungsleiter.stellenNummer = 10014;
87:         herstellungsleiter.geburtsdatum = new GregorianCalendar(1941, 4, 1).getTime();
88:
89:         vertriebsleiter.name = "Life";
90:         vertriebsleiter.vorname = "Deep";
91:         vertriebsleiter.stellenNummer = 10042;
92:         vertriebsleiter.geburtsdatum = new GregorianCalendar(1942, 12, 30).getTime();
93:
94:         kartendesigner.name = "Bumskopp";
95:         kartendesigner.vorname = "Dankward";
96:         kartendesigner.stellenNummer = 10321;
97:         kartendesigner.geburtsdatum = new GregorianCalendar(1954, 3, 21).getTime();
98:
99:         katalogdesigner.name = "Bunker";
100:        katalogdesigner.vorname = "Archie";
101:        katalogdesigner.stellenNummer = 10101;
102:        katalogdesigner.geburtsdatum = new GregorianCalendar(1900, 8, 15).getTime();
103:
104:    }
105:
106: }
```

### Lösungsvorschlag zu Aufgabe 5-6 (a)

```
1: /* $Id: DegreesMinutes.java $
2:  */
3:
4:
5: public class DegreesMinutes {
6:
7:     protected double degrees = Double.NaN;
8:
9:     protected double minutes = Double.NaN;
10:
11: }
```

### Lösungsvorschlag zu Aufgabe 5-6 (c)

```
1: /* $Id: MyGeographicPosition.java $
2:  */
3:
4:
5: public class MyGeographicPosition extends GeographicPosition {
6:
7:     public MyGeographicPosition (double lambda, double phi) {
8:         super(lambda, phi);
9:     }
10:
11:     public MyGeographicPosition (
12:             double lambdaDegrees, double lambdaMinutes,
13:             double phiDegrees, double phiMinutes) {
14:         super(lambdaDegrees, lambdaMinutes, phiDegrees, phiMinutes);
15:     }
16:
17:     public DegreesMinutes lambdaDegreesMinutes () {
18:         DegreesMinutes result = new DegreesMinutes();
19:         result.degrees = (int)super.y;
20:         result.minutes = (super.y - result.degrees) * 60.0d;
21:         return result;
22:     }
23:
24:     public DegreesMinutes phiDegreesMinutes () {
25:         DegreesMinutes result = new DegreesMinutes();
26:         result.degrees = (int)super.x;
27:         result.minutes = (super.x - result.degrees) * 60.0d;
28:         return result;
29:     }
30:
31: }
```

## Aufgabe 5-7

(Übungsblatt 5)

Sind folgende Java-Anweisungen syntaktisch korrekt (mit den Variablen `a` und `b` deklariert mit Typ `int`)? Begründen Sie Ihre Antwort.

a. `a + 1 = b;`
b. `boolean y = true - false;`
c. `b = 1 & 2;`
d. `int a = 'A';`
e. `byte x = (byte)280;`
f. `boolean z = (false == (10 >= 20));`

### Lösungsvorschlag

zu den Teilaufgaben:

a. nicht korrekt: `a + 1` ergibt einen Wert; auf der linken Seite einer Zuweisung (`=`) sind jedoch nur Variablen erlaubt
b. nicht korrekt: der Subtraktions-Operator `-` kann nicht auf den Typ `boolean` angewandt werden
c. korrekt: der „vollständige“ (Bit-weise) Und-Operator kann auf alle primitiven Typen angewandt werden (`b` erhält hier den Wert `3`)
d. nicht korrekt: laut Aufgabenstellung ist bereits eine Variable `a` deklariert, Variablen müssen aber (nur) genau einmal deklariert werden
(*Anmerkung:* wäre noch keine andere Variable `a` deklariert, würde diese Anweisung so korrekt sein und `a` den Wert `65` zuweisen)
e. korrekt: obwohl der Wert `280` außerhalb des Wertebereichs des Typs `byte` liegt, ist diese Anweisung aufgrund des ausdrücklichen Type-Casts `(byte)` korrekt (`x` erhält hier den Wert `24` wegen des Bereichsüberlaufs)
f. korrekt: Deklarierung und Initialisierung von `z` mit einem booleschen Wert, der Ausdruck `(10 >= 20)` ergibt false, `(false == false)` ergibt `true`

**Anmerkung.** Bei Problemen dieser Art ist das „Code Pad“ in BlueJ äußerst hilfreich. Siehe Unterlagen.

*Arne Johannessen, 28. November 2007*