

## Musterlösung zu Übung 1

### Aufgabe 1

Java ist eine Sprache mit *strengem Typsystem* (*strongly typed language*). Assemblersprachen haben üblicherweise *kein* Typsystem. Worin liegt der Unterschied, und was sind die Vorteile der strengen Typisierung? Gibt es Nachteile?

### Lösung:

Als Programmiersprachen mit strengem Typsystem bezeichnet man Sprachen, welche jedem Ausdruck einen Typ zuordnen und sich an einen Regelsatz bei der Zuweisungen von Ausdrücken an Variablen halten.

Darüber hinaus kann man folgende Unterscheidungen treffen:

- Statische Typsysteme vereinbaren für jede Variable einen Typ und bestimmen zur Übersetzung für jeden Ausdruck einen Typ.
- Dynamische Typsysteme vollziehen dies erst während der Ausführung.

In einer Sprache mit dynamischem Typsystem kann also der Typ des aktuellen Werts einer Variable vom Programmpfad abhängen:

```
if (eingabe < 3)
    x = 3;
else
    x = "Danke schön!";
```

Die Variable x ist in diesem Fall untypisiert.

Zum Beispiel besitzt Java ein statisches Typsystem mit einigen dynamischen Elementen, welche von der Verfügbarkeit von Classcasts herrühren.

In Assembler hingegen werden einzelnen Speicherzellen keinem Typ zugeordnet und es besteht keine Möglichkeit eine fehlerhafte Zuweisung zwischen Typen zu erkennen (z.B. eine 64-bit Fließkommazahl wird als 64-bit Ganzzahl verwendet).

Als Zwischending zwischen fehlenden und strengen Typsystemen kann man noch schwache Typsysteme (*weakly typed languages*) nennen. Dies wandeln Typen automatisch um, wenn der Kontext es erfordert ( $3 + "4"$ ) oder verwenden abgewandelte Definitionen des Begriff "Typs" (Stichwort: Duck-Typing in Ruby).

### Vorteile der strengen statischen Typisierung:

- Erster und wichtigster Vorteil ist die Reduzierung von Fehlermöglichkeiten, welche in eine ausführbare Version des Programms gelangen können. Da der Compiler diese Fehler zur Zeit der Übersetzung mit Hilfe des Typsystems feststellt, kann der Entwickler diese schnell und zielgerecht beseitigen. In Sprachen mit dynamischen Typsystemen, wie Perl oder Python, werden solche Typfehler erst bei der Ausführung des

Programms erkannt. In Sprachen ohne Typsystem gibt es keine Fehlererkennung.

- Ein weiterer Vorteil der Bereitstellung von Typinformationen ist die Möglichkeit für den Compiler diese für Optimierungen des erzeugten Codes zu verwenden.

### Nachteile:

- Von der ökonomischen Perspektive der Programmierung gilt es den Zugewinn an Sicherheit gegen den Aufwand an expliziter Typkennzeichnung aufzuwägen. Dies ist der Grund weshalb Skriptsprachen oftmals keine strengen Typsysteme aufweisen, weil bei ihnen die Entwicklungsgeschwindigkeit im Vordergrund steht.

### Aufgabe 2

1. Geben Sie 3 Beispiele für *Ausdrücke* in Haskell an, die jeweils von unterschiedlichem Typ sind!
  - Primitive Typen: 3 - Ganzzahl, 'a' - Zeichen
  - $\backslash x \rightarrow x * x$  - Funktion (Int  $\rightarrow$  Int)
  - (a, b) - Tupel / tuple
  - [a, b] - Liste / list
2. Desgl. in Java!
  - Primitive Typen: 3 - int, 3.0f float
  - Math.sin(3.2) - Methodenaufruf mit Rückgabewert (ohne Rückgabewert ist es kein Ausdruck)
  - new Integer() - Integer
  - new double[3][3] - double[][]
3. Geben Sie 3 Beispiele für *Anweisungen* in Java an!
  - Zuweisung: a = 3;
  - Methodenaufrufe: a();
  - Wertrückgabe: return a;
  - Leere Anweisung: ;

### Aufgabe 3

a) x und y seien vom gleichen Typ. Welcher Typ muss das sein, damit möglichst viele der folgenden Ausdrücke korrekt sind? Welche Werte haben diese korrekten Ausdrücke? Welche Ausdrücke sind in keinem Fall korrekt?

### Lösung:

Zuerst sollten wir uns in Erinnerung rufen, was "x=y" bewirkt. Wir können dies [in der Sprachdefinition nachschlagen](#):

The result of the first operand of an assignment operator must be a variable, or a compile-time error occurs. This operand may be a named variable, such as a local variable or a field of the current object or class, or it may be a computed variable, as can result from a field access (§15.11) or an array access

(§15.13). The type of the assignment expression is the type of the variable after capture conversion (§5.1.10).

At run time, the result of the assignment expression is the value of the variable after the assignment has occurred. The result of an assignment expression is not itself a variable.

Anschließend müssen wir dann noch nachsehen, welcher Operator den Vorzug bekommt (precedence), wenn keine Klammern angegeben sind. Ergebnis: == hat höhere Bindestärke als =.

<code>x==y=x</code>	Auf jeden Fall ungültig, <code>x==y</code> wird zuerst ausgewertet und ergibt einen boolschen Wert. Da dieser keine Variable ist, gibt es einen Fehler bei der Übersetzung.
<code>(x==y)=x</code>	Auf jeden Fall ungültig, weil links ein Ausdruck steht der keine Variable ergibt. Letztlich der gleiche Fall wie 1. aber explizit.
<code>x==(y=x)</code>	Für jeden Typ gültig. Wert: true
<code>x=y==x</code>	Gültig nur für den Typ boolean oder Boolean
<code>(x=y)==x</code>	Für jeden Typ gültig. Wert: true (da x nach der Zuweisung auf beiden Seiten den gleichen Wert hat)
<code>x=(y==x)</code>	Nur gültig für Typ boolean oder Boolean. Wert: <code>y==x</code>

Die meisten Ausdrücke werden also korrekt, wenn x und y boolsche Variablen sind.

b) Gegeben sei die folgende Klasse:

```
class Int {
    static int n;
    int i;
    Int() { i += n++; }
}
```

Welchen Wert hat der Ausdruck `new Int().n + new Int().i`?

**Lösung:**

Es ist gut sich in Erinnerung zu rufen, dass Ausdrücke in Java von links nach rechts ausgewertet und dass Integervariablen mit 0 initialisiert werden. Zur besseren Identifizierung wollen wir den linken Summanden mit L und den rechten Summanden mit R bezeichnen. Die Variable n ist statisch und damit nicht an L oder R, sondern an die Klasse Int gebunden.

<code>new Int()</code>	Vorher: <code>n == 0</code> , <code>L.i == 0</code> , Nachher: <code>n == 1</code> , <code>L.ii == 0</code> (beachte <code>n++</code> ist ein Postinkrement).
<code>.n</code>	Der links Summand bekommt den Wert 1.
<code>new Int()</code>	Vorher: <code>n == 1</code> , <code>R.i == 0</code> , Nachher: <code>n == 2</code> , <code>R.ii == 1</code> .
<code>.i</code>	Der rechte Summand bekommt den Wert 1
<code>+</code>	Die Summe wird berechnet. Ergebnis: 2.

## Aufgabe 4

Eine Methode `contains` soll prüfen, ob eine Zahl `x` in einem Feld `a` vorkommt. Wir betrachten zwei Versionen von `contains`:

```
boolean contains(int[] a, int x) {
    for(int y: a)
        if(x==y) return true;
        else    return false;
}
boolean contains(int[] a, int x) {
    try { for(int i=0;; i++)
        if(x==a[i]) return true; }
    catch(Exception e) { return false; }
}
```

Richtig oder falsch? Bitte genauer erläutern und begründen!

### Lösung:

Die erste Version ist falsch, da sie

- 1.) die Rückgabe von `false` an der falschen Stelle erfolgt. Macht euch klar, dass damit bereits bei der Überprüfung des ersten Feldelements entweder `true` oder `false` zurückgegeben wird. Weitere Elemente werden nie betrachtet. Korrektur diesen Problems:

```
boolean contains(int[] a, int x) {
    for(int y: a)
        if(x==y) return true;
    return false;
}
```

- 2.) nicht berücksichtigt, dass der Variablen `a` auch der Wert `null` übergeben werden kann. In diesem Fall würde eine `NullPointerException` in der `for`-Schleife geworfen werden. Alternativ könnte man aber auch sagen, dass sich Version eins bezüglich dieser Besonderheit korrekt verhält, dann ist Version zwei falsch.

Die zweite Version ist korrekt. Allerdings gilt es zu bedenken, dass diese Version erst ab einer Größe von ca. 1000 Elementen für das Feld einen Leistungsvorsprung durch den fehlenden Vergleich ergibt.

Es lässt sich allerdings darüber streiten, ob ein solcher Einsatz im Hinblick auf Verständlichkeit und Lesbarkeit (, ohne welche der Quelltext oftmals wesentlich fehleranfälliger ist,) gerechtfertigt ist.

## Aufgabe 5:

Vervollständigen Sie die Klasse `Queue` aus der Vorlesung, 1.1-14/15, d.h. mittels eines Geflechts.

### Lösung:

Ihr findet ein Jar-File mit der Lösung unter <http://www.inf.fu-berlin.de/lehre/WS05/ALP3/loesungen/U1.A5.jar>. Das Jar-File enthält auch einen JUnit-Testfall, mit dem Ihr eure Implementierung ein paar einfachen Tests unterziehen könnt. Hier nur der neue Teil der Klasse `queue`:

```
public Item remove() throws Exception {
    if (front == null)
        throw new NoSuchElementException();
    Item result = front.value;
    front = front.next;
    if (front == null)
        rear = null;
    return result;
}

public int length() {
    int result = 0;
    Cell current = front;
    while (current != null) {
        result++;
        current = current.next;
    }
    return result;
}
```