

Fragen, Antworten, Kommentare zur aktuellen Vorlesung

Hinweis: Die Diskussion einer Beispiellösung zur Aufgabe 11 zu Fachbegriffen am Beispiel der Klasse Konto finden Sie unter <https://youtu.be/WvitFQCo8b4> (19:59).

Frage: Ich habe in anderen Videos gesehen, dass Eclipse genutzt wird und zum Starten eine besondere Methode benötigt wird. Ist diese Methode auch in BlueJ nutzbar?

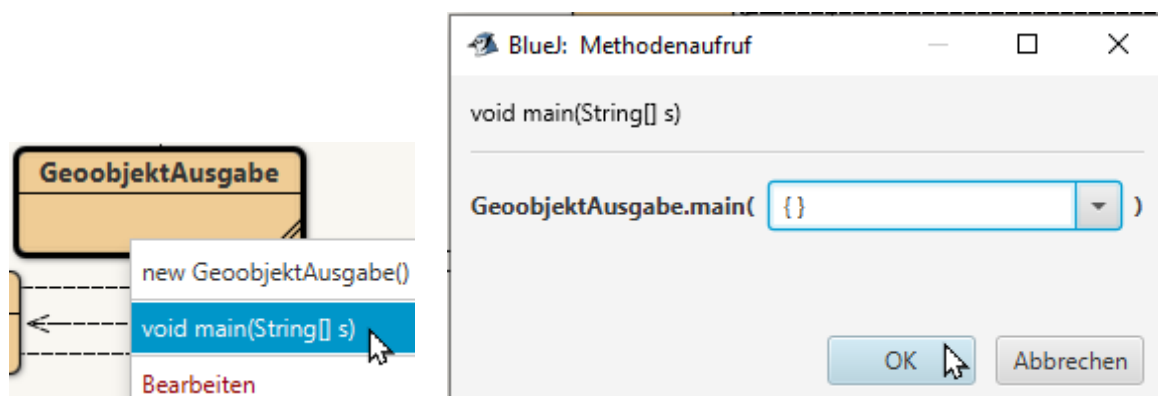
Generell ist dies auch in BlueJ machbar, hier zusätzlich die Argumente, warum wir uns das noch nicht ansehen. In der Vorlesung ist es typisch, dass zum Ausprobieren der Lösung z. B. eine Klasse Analyse geschrieben wird, die dann eine Methode `public void mach() {...}` enthält, in der dann alle Experimente stattfinden. Um dies in Eclipse nutzen zu können, wird eine Methode der folgenden Form benötigt.

```
public static void main(String[] args){
    Analyse ana = new Analyse();
    ana.mach();
}
```

Danach kann das Programm über einen Run-Button gestartet werden. Der generelle Nachteil ist zunächst, dass man in Eclipse nicht einfach mal einen Konstruktor ausführt, sich das Objekt inhaltlich anschauen und Methoden darauf ausführen kann. Das mir sehr wichtige Experimentieren ist für Personen, die die Programmierung gerade beginnen, nicht möglich.

Ein zweiter wichtiger KO-Punkt ist, dass die anzugebende Methode für Anfänger nach Voodoo aussieht, man muss es halt hinschreiben. Die Vorstellung, dass man was hinschreibt und irgendwas passiert und ein Programm dann funktioniert, ist für unerfahrene Personen sehr gefährlich, da sie dann nicht systematisch nach Lösungen suchen, sondern vermuten, dass ihnen ein Voodoo-Schritt fehlt.

Wir lernen die Methode später kennen, dann wissen wir aber vorher, was `static` und `String[]` ist. Gerade `static` ist bei Methoden eine Besonderheit, die bei häufigerer Verwendung bei unerfahrenen Personen meist auf das mangelnde Verständnis von Objektorientierung hinweist.



Das Bild auf der linken Seite zeigt, dass die `main()`-Methode direkt auf der Klasse in der die Methode steht, ausgeführt werden kann. Das Bild daneben zeigt, dass als Parameter vereinfachend der bereits vorgeschlagene Wert `{}` (leeres Array, kommt später) zum Starten übergeben wird.

Frage: Ich bekomme folgende merkwürdige Fehlermeldung, was kann ich machen?

```
public static void main(String[] args) {  
    new GeoobjektAusgabe().dialog();  
}
```

illegal character: '\u0000'

Das kann passieren, wenn Programmfragmente mit Copy&Paste aus Word-Dokumenten, PDF-Seiten, Web-Seiten oder ähnliches kopiert werden. Generell deutet \u an, dass es sich um ein Unicode-Zeichen handelt. Hier ein nicht sichtbares Zeichen, das wie ein Leerzeichen ausgegeben wird, aber von Java als Teil eines Befehls angesehen wird. Die Lösung ist alle vermeintlichen Leerzeichen (oftmals ein Tabulator) in der Umgebung des Fehlers zu löschen. Bei weitergehenden Problemen kann es sinnvoll sein, dass kopierte Teilstück erst in einen einfachen Editor zu kopieren und danach erneut nach BlueJ zu kopieren.

Dieser Fehler deutet auf ein Feature hin, dass Variablenamen fast beliebige Unicode-Zeichen enthalten können, es ist also auch folgendes möglich, aber für uns völlig sinnlos. Ein Sinn der Möglichkeit besteht darin, dass Menschen die andere Zeichensätze gewohnt sind, Variablen in ihren eigenen Zeichensätzen schreiben können, was gerade am Anfang der Programmierausbildung sinnvoll ist. Der Editor muss diese Möglichkeit dann unterstützen, was in BlueJ bei Variablenamen nicht der Fall ist. Folgendes funktioniert aber:

```
String text = "😊";  
int \u1F602 = 42;  
int Δ = 1;  
double π = 3.141592;  
String 你好 = "hallo";  
Δ = Δ + Δ;
```

Später ist es leider meist notwendig, dies auf „imperialistische“ ASCII-Zeichen zu reduzieren.

Frage: Bei der Formularaufgabe mussten viele add(.)-Aufrufe auf dem gleichen Objekt gemacht werden, das finde ich recht aufwändig. Im Internet habe ich den {}-Operator gefunden, können wir den auch nutzen?

Antwort: Einige Bemerkungen vorweg: Richtig, die mehreren add(.) sind etwas lästig, aber wir kennen bis jetzt noch keine Abkürzungen. Generell gilt, dass sich alle Aufgaben immer ausschließlich mit dem Vorlesungsinhalt vollständig bearbeiten lassen. Wenn Sie nach dem Finden der aus der Vorlesung hervorgehenden Lösung, wie hier, nach Neuem in Internet suchen und experimentieren ist das aber sinnvoll und es können gerne Fragen wie diese dazu gestellt werden. Trotzdem hier die Warnung, die genauen Details der Lösung, gehen weit, sehr weit, wirklich sehr sehr weit über den Inhalt der Grundlagenvorlesung hinaus. Ich erwarte im Gegensatz zum Vorlesungsinhalt nicht, dass dies vollständig verstanden wird.

Der dahinter liegende Ansatz soll durch folgendes Programm verdeutlicht werden.

```
import java.util.ArrayList;
```

```

public class MainInitialisierung {

    public void main() {
        EinUndAusgabe io = new EinUndAusgabe();
        ArrayList<String> eso = new ArrayList<>() {
            {
                add("Hai");
                add("fisch");
            }
        };
        ArrayList<String> liste = new ArrayList<>();
        liste.add("keine");
        liste.add("Ueberraschung");
        io.ausgeben(eso + "\n");
        io.ausgeben(liste + "\n");
        io.ausgeben(io.getClass() + "\n");
        io.ausgeben(eso.getClass() + "\n");
        io.ausgeben(liste.getClass() + "\n");

        EsoPunkt p = new EsoPunkt();
        p.ausgeben(io);
        p.setY(100);
        p.ausgeben(io);
    }
}

```

Wird der gelbe Block anders formatiert, stehen da doppelt geschweifte Klammern nebeneinander. Die obige Formatierung sollte aber schon deutlich machen, dass es sich nicht um einen Operator sondern nur um normale Blockklammern handelt.

Was passiert da genauer? Es sieht zunächst nach einem normalen Konstruktoraufruf mit `new ArrayList<>()` aus, ist es aber nicht, da geschweifte Klammern direkt als Block dahinter folgen. Dadurch wird eine neue Klasse im Code deklariert, die von `ArrayList` erbt (später in der Vorlesung). Dies macht auch die Ausgabe der ersten 5 `io`-Ausgaben in den ersten 5 Zeilen etwas deutlich.

```

[Hai, fisch]
[keine, Ueberraschung]
class EinUndAusgabe
class MainInitialisierung$1
class java.util.ArrayList
42 : 43
42 : 100

```

Es ist zu erkennen, dass zunächst die Listen normal ausgegeben werden. Dann wird die von der Klasse `Object` (später in der Vorlesung) geerbte Methode `getClass()` genutzt, die zu einem Objekt ein sogenanntes Klassenobjekt liefert, dass u. a. in der Ausgabe einfach seinen Namen als Ergebnis hat. Die Ausgabe zeigt, dass es sich bei `eso` nicht um eine `ArrayList`, sondern um ein sogenanntes anonymes Objekt der im Code angegebenen Klasse handelt, das zur Laufzeit des Programms entstanden ist.

Es bleibt zu klären, was mit den inneren geschweiften Klammern passiert. Dies ist ein Initialisierungsblock, der in jeder Klasse stehen kann und typischerweise vor jeder Objekterzeugung ausgeführt wird. Das soll mit dem zweiten Teil des Programms und der fehlenden Klasse `EsoPunkt` verdeutlicht werden.

```

public class EsoPunkt {
    private int x;

```

```

private int y;
{ // nie, nie in dieser Vorlesung über die Nutzung nachdenken (nie)
  this.x = 42;
  this.y = 41;
}

public EsoPunkt() {
  this.y = 43;
}

public void setY(int val) {
  this.y = val;
}

public void ausgeben(EinUndAusgabe io) {
  io.ausgeben(this.x + " : " + this.y + "\n");
}
}

```

Die bereits vorher gezeigte Ausgabe macht deutlich, dass zunächst der gelbe Block und dann der Konstruktor ausgeführt wird. Es stellt sich die Frage, warum so ein Block in fast jedem einführenden Java-Buch fehlt. Der Grund ist, dass es nicht garantiert werden muss, dass dieser Block ausgeführt wird. Es gibt Java-Frameworks, die das bewusst verhindern, da stattdessen deren interner Code ausgeführt wird. Da die Ausführung des Konstruktors garantiert ist, wird immer mit diesem in die Objekterzeugung eingestiegen.

Die Programmierung ist nebenbei ein Beispiel für einen esoterischen Programmierstil, der gegen die elementare Grundregel verstößt, dass Programme so einfach lesbar wie möglich geschrieben sein müssen. Solche Leute fliegen nach dem ersten Code-Review in der Probezeit raus; Studierende müssen nur ohne Abendessen ins Bett.

Das Beispiel sollte deutlich zeigen, dass bei es bei Unsicherheiten keine gute Idee ist einfach mal Google zu befragen und auf eine Lösung zu hoffen. Sie müssen bei jedem Programmierschritt genau wissen, was passiert und was sie machen wollen, ansonsten kann Ihnen der Einstieg in die Software-Entwicklung nicht gelingen. Nutzen Sie die vielfältigen Möglichkeiten zum Nachfragen.