

## Übungen zu Einführung in die Informatik I

### Erste Schritte mit Java

Im weiteren Verlauf der Übungen zu „Einführung in die Informatik I“ soll die Programmiersprache Java verwendet werden. Dazu steht das Java Development Kit (JDK) der Version 1.2 an den Sun-Rechnern in der Informatikhalle zur Verfügung (für die in den Übungen entwickelten Programme ist auch das JDK 1.1 ausreichend). Mit diesem ist es möglich, Java-Programme zu übersetzen, ablaufen zu lassen, nach Fehlern zu suchen usw. Dieses Arbeitsblatt ist nur dazu gedacht, die ersten Schritte mit Java zu erleichtern; keineswegs will und kann es eine vollständige Beschreibung der Java-Programmierung liefern.

#### **Implementieren eines Java-Programms:**

Wir folgen hier dem in der Vorlesung eingeführten Schema mit den folgenden Aktivitäten:

- a) **Erstellen** bzw. **Editieren** einer Java-Quelldatei *Name*.java: Diese Datei enthält eine oder mehrere Klassendefinitionen; aus Gründen der Übersichtlichkeit ist es sinnvoll, in einer Datei jeweils nur eine einzige Klasse zu definieren und Klassennamen sowie Dateinamen gleich zu wählen. Konvention für Klassenbezeichner ist, dass der erste Buchstabe groß geschrieben wird. Im Weiteren gehen wir von dieser Namenskonvention aus.

Eine Klassendefinition sieht dabei folgendermaßen aus:

```
class Name {  
    Klassenbeschreibung  
}
```

Der Aufbau einer Java-Quelldatei wird später noch näher erläutert.

- b) **Compilieren** (Übersetzen) der Java-Quelldatei: Dies geschieht durch Eingabe des Befehls

```
javac Name.java
```

in der Unix-Kommando-Shell. Für eine Datei *ErstesProgramm.java* lautet der Befehl zur Übersetzung also:

```
javac ErstesProgramm.java
```

Der Java-Übersetzer (Compiler) generiert aus der Quelldatei eine Datei namens *Name.class*. Bricht der Übersetzer mit Fehlermeldungen ab, so konnte aus der Quelldatei aufgrund von Fehlern keine Datei *Name.class* erzeugt werden. *Bemerkung:* Als Alternative zu dem Übersetzer `javac` können Sie in der Rechnerhalle auch das Programm `jikes` verwenden (dieses gibt besser verständliche Fehlermeldungen aus und arbeitet etwas schneller).

c) **Exekutieren** (Ausführen) des Programmes: Durch die Eingabe des Befehls

```
java Name
```

wird das Java-Programm gestartet. Sollen dem Programm Parameter übergeben werden, so sind diese durch Leerzeichen getrennt hinter `java Name` anzufügen.

Also sähe ein Programmaufruf von ErstesProgramm mit dem Wert 3 so aus:

```
java ErstesProgramm 3
```

## Aufbau einer Java-Quelldatei:

Eine Java-Quelldatei besteht aus ein (oder mehreren, s.o) Klassendefinitionen.

Eine Klassendefinition sieht dabei folgendermaßen aus:

```
class Name {  
  
    Variablen- und Methodenvereinbarungen (Deklarationen)  
  
}
```

Nach der zuvor erwähnten Konvention soll die Java-Datei *Name.java* heißen, wenn *Name* der Bezeichner der Klasse ist.

Variablen werden folgendermaßen deklariert:

```
Variablentyp Variablenname ;
```

*Variablentyp* gibt die Art einer Variable an, *Variablenname* den gewünschten Bezeichner.

Beispielhaft einige Variablendeklarationen für eine Beispielklasse `EineKlasse` :

```
class EineKlasse {  
  
    int ganzeZahl;  
    boolean kleiner;  
    char einBuchstabe;  
    double naeherungswert;  
    MeineKlasse meinObjekt;  
  
}
```

Jedes Objekt der Klasse `EineKlasse` besitzt Attributvariablen für eine ganze Zahl, einen Wahrheitswert, ein Zeichen, eine Fließkommazahl und ein Objekt einer (an anderer Stelle definierten)

Klasse *MeineKlasse*. Diese Beziehung, dass Objekte einer Klasse *Name* andere Komponenten beinhalten, wird oft auch als **Aggregation** bezeichnet. In der graphischen Notation UML (*Unified Modeling Language*) wird dies in Klassendiagrammen durch eine Beziehung, die an der beinhaltenen Klasse mit einer Raute beginnt, beschrieben:



Abbildung 1: UML–Diagramm: Aggregationsbeziehung zwischen Klassen

Neben der Aggregationsbeziehung ist in objekt–orientierten Programmiersprachen wie Java auch die so genannte **Vererbung** von großer Bedeutung. Man kann eine Klasse als **Unterklasse** einer anderen Klasse definieren und verleiht damit der Unterklasse die Eigenschaften und Fähigkeiten der **Oberklasse**. Objekte der Unterklasse verfügen damit auch über die **Merkmale**, d.h. alle Attribute und Operationen, die Objekte der Oberklasse besitzen. In Java wird diese Vererbungsbeziehung in der Definition der Unterklasse durch das Schlüsselwort `extends` ausgedrückt:

```

class Unterklasse extends Oberklasse {
    ...
}
  
```

In UML wird diese Beziehung zwischen Klassen durch einen Pfeil mit hohler Dreiecksspitze von der Unterklasse zur Oberklasse ausgedrückt:

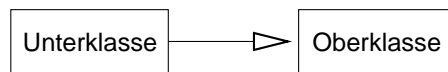


Abbildung 2: UML–Diagramm: Vererbungsbeziehung zwischen Klassen

In objekt–orientierten Sprachen werden Operationen meist als **Methoden** bezeichnet. Eine Methode wird so vereinbart:

```

Ergebnistyp Methodenname (Parameterliste) {
    Methodenrumpf
}
  
```

*Ergebnistyp* sei wiederum die Art des Rückgabergebnisses (manche Methoden haben kein Rückgabergebnis, was in Java durch das Schlüsselwort `void` gekennzeichnet wird),

*Methodenname* der selbstgewählte Bezeichner der Methode,

*Parameterliste* eine durch „`,`“ getrennte Folge von Variablen mit Typangabe und

*Methodenrumpf* eine Folge von Anweisungen.

Die Kopfzeile einer Methodenvereinbarung ist eng verwandt mit der in der Vorlesung definierten Funktionalität einer Operation.

Hier einige einfache Beispiele von Methoden innerhalb einer Klasse *Info1Klausur*:

```

class Info1Klausur {
  
```

```

String semester;

boolean istBestanden (int punktzahl) {
    return (punktzahl >= 17);
}

String gibSemester () {
    return semester;
}
}

```

Das Schlüsselwort `return` signalisiert, dass die Methode den danach folgenden Ausdruck als Ergebnis zurückliefert.

Die in der Klasse `Info1Klausur` definierten Methoden können bei der Ausführung eines Programms aufgerufen werden. Die Klasse `Info1Klausur` selbst stellt allerdings kein eigenständig lauffähiges Programm dar. Auf Arbeitsblatt 2 werden wir behandeln, wie in Java eigenständig lauffähige Programme (Anwendungen) erstellt werden.

## Klassen und Objekte in Java

Objekte werden in Java durch die Instantiierung einer Klasse erzeugt. Wie in der Vorlesung definiert, ist ein **Objekt** eine **Instanz** einer Klasse, wenn es Element der Menge aller Objekte dieser Klasse ist. Für die Erzeugung einer Instanz wird in Java ein so genannter **Konstruktor** verwendet. Dieser ist eine Operation einer Klasse, die den Namen der Klasse selbst trägt. Würde man das Beispiel eines Kellers natürlicher Zahlen  $\text{Keller}(\mathbb{N})$  aus der Vorlesung auf Java übertragen, so würde die Erzeugung eines Objektes der Klasse  $\text{Keller}(\mathbb{N})$  durch die Operation

$$\text{create: } \rightarrow \text{Keller}(\mathbb{N})$$

umgesetzt werden in einen Konstruktor `KellerNat()` innerhalb der Klasse `KellerNat`:

```

class KellerNat {

    KellerNat() {}

    ... andere Operationen push, pop, top
}

```

Die Erzeugung eines Objektes einer Klasse findet dann unter Verwendung des Schlüsselworts `new` verbunden mit einer Konstruktor-Operation statt. In einem Programm würde also durch

$$\text{new KellerNat}();$$

eine Instanz der Klasse `KellerNat` generiert werden.

Eine genauere Behandlung der Instanzerzeugung und von Konstruktoren wird im Laufe der Übungen und im Vorlesungskapitel „Objekt-orientierte Programmierung“ erfolgen.

### Literatur zu Java:

Ale Literatur zur Programmiersprache Java verweisen wir auf der WWW-Seite zu den Übungen <http://www.bruegge.informatik.tu-muenchen.de/teaching/ws00/Info1/uebungen.html> auf einige, für TU-Studenten kostenlose, Online-Dokumentationen.