Ludwig-Maximilians-Universität München
Institut für Informatik
PD. Dr. Peer Kröger, Dr. Andreas Züfle
Daniel Kaltenthaler, Klaus Schmid

München, 13.01.2015

Einführung in die Programmierung

WS 2014/15

Übungsblatt 12: Vererbung, Polymorphismus

Besprechung: 21./23./26.01.2015

Ende der Abgabefrist: Dienstag, 20.01.2015 14:00 Uhr.

Hinweise zur Abgabe:
Geben Sie bitte Ihre gesammelten Lösungen zu diesem Übungsblatt in einer Datei `loesung12.zip` unter `https://uniworx.ifi.lmu.de` ab. Alle Code-Aufgaben müssen als .java abgegeben werden. PDF-code wird nicht akzeptiert. Nicht kompilierbare Abgaben werden mit 0 Punkten bewertet.

Aufgabe 12-1 *Vererbung* **4+0+0 Punkte**

Es gibt Schiffe und als Spezialfälle davon Segelschiffe und Motorschiffe. Jedes Schiff hat eine Tonnage, jedes Segelschiff hat zusätzlich zur Tonnage eine Segelfläche und jedes Motorschiff hat zusätzlich zur Tonnage eine Motorleistung. Segelschiffe haben keine Motorleistung und Motorschiffe haben keine Segelfläche, andere Schiffe haben weder das eine noch das andere.

(a) Definieren Sie eine Klasse `Schiff` und zwei Unterklassen `Segelschiff` und `Motorschiff`. Die Datei `Schifffahrt.java`, die Sie auf der Homepage finden, enthält ein Hauptprogramm, das die drei zu definierenden Klassen verwendet, das aber für diese Teilaufgabe nicht verändert werden soll.

Tonnage, Motorleistung und Segelfläche sollen durch **`private`** Attribute (Instanzvariablen) der entsprechenden Klassen repräsentiert sein, deren Werte als Parameter der Konstruktoren mitgeliefert werden. Zur Implementierung der Konstruktoren ist es in einigen Fällen sinnvoll, **`super`**`(...)` zu verwenden.

Jede der drei Klassen soll eine Methode `toString()` definieren, die eine Textdarstellung des jeweiligen Objekts liefert. Das Format der Textdarstellung entnehmen Sie den Kommentaren hinter den Ausgabeanweisungen in `Schifffahrt.java`. Auch bei der Implementierung dieser Methoden ist **`super`** sinnvoll, aber diesmal ohne Klammern.

(b) Schließen Sie die Deklarationen der Methode `toString()` in allen drei Klassen in `/*` und `*/` ein, so dass sie vorübergehend entfernt werden. Das Programm bleibt trotzdem übersetzbar und ablauffähig. Erklären Sie, warum das so ist und welcher Unterschied zu der Version mit den Methoden besteht. Schreiben Sie diese Erklärung als Kommentar hinter das Hauptprogramm in die Datei `Schifffahrt.java`.

(c) Reaktivieren Sie Ihre Deklarationen der Methode `toString()` und übersetzen Sie alle Dateien neu. Starten Sie das Programm einmal mit `java Schifffahrt` und einmal mit `java Schifffahrt xyz` und vergleichen Sie die Ausgabe für `c`.

Erklären Sie anhand dieses Beispiels, warum ein Java-Compiler im allgemeinen nicht feststellen kann (zur Compile-Zeit), welcher Java-Programmcode bei einem Methodenaufruf ausgeführt werden muss. Schreiben Sie diese Erklärung ebenfalls als Kommentar ans Ende der Datei `Schifffahrt.java`.

Geben Sie die Klassen `Schiff`, `Segelschiff` und `Motorschiff` sowie die Datei `Schifffahrt.java` mit den Kommentaren für (b) und (c) ab.

Aufgabe 12-2 *Polymorphismus* **8 Punkte**

Gegeben sind die untenstehenden Klassen `A`, `B` und `C`.

Geben Sie in einer Datei `Polymorphismus.txt` oder `Polymorphismus.pdf` an, welche Ausgabe bei Ausführung der Klasse `C` zu erwarten ist und beschreiben Sie, welche Methoden ausgeführt werden, auf welche Variablen zugegriffen wird und warum.

```
public class A {

  private int a;

  public A(int a){
    this.a = a;
  }

  public int getA(){
    return this.a;
  }

  public int add(A a){
    return this.a + a.getA();
  }

  public int m(){
    return 1;
  }

  public int n(){
    return this.m();
  }

  public String toString(){
    return "Ich bin ein A.";
  }
}
```

```
public class B extends A {

  private int a;

  public B(int a){
    super(a);
  }

  public int getA(){
    return this.a;
  }

  public int add(B b){
    return this.getA() + b.getA();
  }

  public int m(){
    return 2;
  }

  public String toString(){
   return "Ich bin ein B.";
  }
}
```

```
1  public class C {
2
3    public static String print(A a){
4      return "Parameter A: " + a.toString();
5    }
6
7    public static String print(B b){
8      return "Parameter B: " + b.toString();
9    }
10
11   public static void main(String[] args) {
12
13     A o1 = new A(2);
14     B o2 = new B(3);
15     A o3 = new B(4);
16
17     System.out.println(o1.m() + o2.m());
18     System.out.println(o2.m() + o3.m());
19
20     System.out.println(o2.n());
21
22     System.out.println(print(o1));
23     System.out.println(print(o2));
24     System.out.println(print(o3));
25
26     System.out.println(o1.add(o1));
27     System.out.println(o1.add(o2));
28     System.out.println(o2.add(o3));
29     System.out.println(o2.add(o2));
30   }
31 }
```

Aufgabe 12-3 *Interfaces und Ausnahmen* **0 Punkte**

Eine Matrix ist ein rechteckiges Schema von Zahlen oder anderen Größen, die addiert und multipliziert werden können. Viele von Ihnen kennen das Rechnen mit Matrizen aus der linearen Algebra, für die anderen weisen wir kurz auf die Eigenschaften hin, die wir hier benötigen.

Wenn Matrizen von der Größe her zusammenpassen, ist es möglich, sie zu addieren oder miteinander zu multiplizieren. Das folgende Beispiel

$$A := \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}$$

zeigt eine Matrix A mit drei Zeilen und vier Spalten. Allgemein spricht man von einer $m \times n$-Matrix mit m Zeilen und n Spalten. Die Werte a_{ij} entsprechen dem Wert der Matrix in der i-ten Zeile und j-ten Spalte.
Hier benötigen wir folgende Rechenvorschriften:
Die Summe zweier $m \times n$-Matrizen A und B wird berechnet durch stellenweise Addition der Einträge der beiden Matrizen:

$$A + B := (a_{ij} + b_{ij})_{i=1,\ldots,m;j=1,\ldots,n}$$

Beispiel:

$$\begin{pmatrix} 2 & 5 & 4 \\ 1 & 2 & 3 \end{pmatrix} + \begin{pmatrix} 6 & 2 & 0 \\ 1 & 3 & 4 \end{pmatrix} = \begin{pmatrix} 2+6 & 5+2 & 4+0 \\ 1+1 & 2+3 & 3+4 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 4 \\ 2 & 5 & 7 \end{pmatrix}$$

Es können nur Matrizen mit der gleichen Anzahl an Zeilen und der gleichen Anzahl an Spalten addiert werden.
Eine Matrix A wird mit einem Skalar λ multipliziert, indem alle Einträge der Matrix mit dem Skalar multipliziert werden:

$$\lambda * A = (\lambda * a_{ij})_{i=1,\ldots,m;j=1,\ldots,n}$$

Beispiel:

$$3 * \begin{pmatrix} 2 & 5 & 4 \\ 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 3*2 & 3*5 & 3*4 \\ 3*1 & 3*2 & 3*3 \end{pmatrix} = \begin{pmatrix} 6 & 15 & 12 \\ 3 & 6 & 9 \end{pmatrix}$$

Zwei Matrizen $A = (a_{ij})_{i=1,\ldots,m;j=1,\ldots,n}$ und $B = (b_{ij})_{i=1,\ldots,n;j=1,\ldots,o}$ werden folgendermaßen miteinander multipliziert:

$$A * B := \left(\sum_{k=1}^{n} a_{ik} * b_{kj} \right)_{i=1,\ldots,m;j=1,\ldots,o}$$

Das Ergebnis der Multiplikation einer $m \times n$ Matrix mit einer $n \times o$ Matrix ist also eine $m \times o$ Matrix.
Beispiel:

$$\begin{pmatrix} 2 & 5 & 4 \\ 1 & 2 & 3 \end{pmatrix} * \begin{pmatrix} 2 & 5 \\ 1 & 3 \\ 0 & 4 \end{pmatrix} = \begin{pmatrix} 2*2+5*1+4*0 & 2*5+5*3+4*4 \\ 1*2+2*1+3*0 & 1*5+2*3+3*4 \end{pmatrix} = \begin{pmatrix} 9 & 41 \\ 4 & 23 \end{pmatrix}$$

Ganz entsprechende Operationen sind Addition und Multiplikation einer Matrix A mit einem Vektor V bzw. umgekehrt, indem ein Vektor als Matrix mit einer Spalte (bzw. einer Zeile) aufgefasst wird.
Insbesondere kann deshalb die Multiplikation einer $m \times n$ Matrix A mit einem $n \times 1$ Vektor v als Transformation des Vektors in einen anderen Vektorraum aufgefasst werden, da das Ergebnis von $A * v$ ein $m \times 1$ Vektor ist.
Zwei Vektoren v und w gleicher Länge n können durch Transponieren eines der beiden Vektoren multipliziert werden. Fassen wir v als $n \times 1$, w als $1 \times n$ Vektor auf, so wird die Multiplikation $v * w$ *Tensorprodukt* genannt und das Ergebnis ist eine $n \times n$ Matrix. Fassen wir umgekehrt v als $1 \times n$, w als $n \times 1$ Vektor auf, so wird die Multiplikation $v * w$ *kanonisches Skalarprodukt* genannt, das Ergebnis ist eine Zahl.

In dieser Aufgabe sollen zwei Klassen `ReellerVektor` und `ReelleMatrix` zur Darstellung reellwertiger Vektoren und Matrizen programmiert werden. Implementieren Sie dazu die Interfaces `Vektor.java` und `Matrix.java`, die Sie auf der Vorlesungshomepage finden.

Achten Sie darauf, dass Sie auch geeignete Konstruktoren für beide Klassen zur Verfügung stellen, und verwenden Sie in sinnvoller Weise Exceptions, um ungültige Methodenaufrufe zu behandeln (Exceptions werden in der Vorlesung noch behandelt werden).

Hinweis: Die Klasse **java.text.NumberFormat** könnte hilfreich sein.