Arbeitsbogen Nr.6 Java-Kurs

Bodo Werner **WiSe 01/02**

Themen: **Klassen und Objekte (Teil 1)**

Die Schlüsselworte `class` und `new`, die Klasse `KomplexeZahl`, Konstruktoren und ihre Überladung, Daten und Methoden einer Klasse, die Initialisierung (Allokierung) eines Objekts einer Klasse mit Hilfe eines Konstruktors und des `new`-Operators, die Verwendung des Schlüsselwortes `this`.

Zugriffsrechte bzw. Sichtbarkeit von Klassen, daten und Methoden durch die Zusätze `public, private` (und `protected`).
Siehe auch

`http://www.math.uni-hamburg.de/home/werner/JavaKurs/JavaKurs6.html`

Aufgabe 15:

a) Fügen Sie der Applikationsklasse `Komplex2` analog zur Methode `mal()` noch eine Methode

```
public static KomplexeZahl1 addiere(KomplexeZahl1 z1, KomplexeZahl1 z2)
```

zu, die zwei komplexe Zahlen addieren soll. Berechnen Sie mit diesen Methoden das Produkt und die Summe von $z_1 = 3 + 2i$ und $z_2 = -1 - 2i$. (Vorher sollten Sie die Klassen `KomplexeZahl1` und `Komplex2` aus dem Internet laden.) Überprüfen Sie mit Hilfe der Methode `reell()`, welche der beiden Ergebnisse reelle Zahlen darstellen.

b) Die Klasse `KomplexeZahl4` (die die Klasse `Winkel` benutzt) verbietet einen Zugriff von auẞen auf die Daten. Daher kann die Methode `mal()` in der Applikationsklasse `Komplex2` nicht übernommen werden. Erstellen Sie eine Applikationsklasse, in der die Methode `mal()` entsprechend modifiziert wird, indem die öffentlichen Methoden `getRe()` und `getIm()` verwendet werden. Sodann verifiziere man an Hand der beiden komplexen Zahlen mit Betrag 2 bzw. 3 und mit Winkel 1.2π bzw. 1.5π, dass die Multiplikation komplexer Zahlen dem Produkt ihrer Beträge und der Summe ihrer Winkel entspricht.

Zusatzaufgabe (für nicht Ausgelastete): Fügen Sie der Klasse `KomplexeZahl4` eine Methode `void Drehe(Winkel a)` hinzu, die die jeweilige komplexe Zahl um einen vorgebenen Winkel a im Bogenmaß gegen den Uhrzeiger dreht. Hinweis: Drehen um den Winkel a ist identisch mit einer Multiplikation mit $z = \cos(a) + i\sin(a)$.

Aufgabe 16:

Erstellen Sie eine öffentliche Klasse namens `Vektor3`, die als wichtigstes (privates) Attribut ein `double`-Feld (`double[] x`) mit **drei** Komponenten (ein **Vektor** im Raum), aber auch die euklidische Länge (den Betrag) (`r`) des Vektors enthalten soll. Der Konstruktor soll drei `double`-Zahlen übergeben. Erstellen Sie eine öffentliche Methode `getLaenge()` dieser Klasse, die die Länge des Vektors berechnet und die im Konstruktor aufgerufen wird, um das Attribut `r` zu berechnen.

Schreiben Sie eine Applikationsklasse, die (analog zu `Komplex1` oder `Komplex2`) den Vektor (1.1, 2.2, 3.3) konstruiert und dessen euklidische Länge durch Aufruf der Methode `getLaenge()` ausgibt. Warum kann nicht direkt auf das Attribut `r` zugegriffen werden?

Wählen Sie als Bezeichner des `double`-Feldes `x`, als Bezeichner der drei Parameter des Konstruktors `x, y, z`. (Dadurch wird die Verwendung von `this` zwingend.)

Nun fügen Sie der Klasse einen zweiten Konstruktor bei, der nur zwei `double`-Parameter enthält, die den ersten beiden Komponenten des Vektors entsprechen sollen. Die dritte Komponente soll Null gesetzt werden, d.h. man es hat mit Vektoren in der Ebene $z = 0$ zu tun. Konstruieren Sie auf diese Weise den Vektor (1.1, 2.2, 0).

Ein Vektor kann mit einem Skalar komponentenweise multipliziert werden. Fügen sie der Klasse `Vektor3` eine Methode `public void skalarMultiplikation(double a)` hinzu, die den Vektor durch komponentenweise Multiplikation mit `a` verändert.
Testen Sie diese Methode in einer Applikationsklasse, indem Sie den Vektor (1.1, 2.2, 0) mit $a = 2$ skalar multiplizieren und die Komponenten des Ergebnisses ausgeben.