Übungsaufgabenserie 12
Grundlagen der Informatik und Numerik
Abgabe: 22. 01. 2014, 23:55 Uhr, elektronisch

1. Tangens.java, Cotangens.java, TanCotVergleich.java

Entwickeln Sie zur Erweiterung der mathematischen Funktionen aus der Vorlesung je eine Klasse `Tangens` und `Cotangens`. Diese Klassen sollen das Berechnen der Funktionswerte von tan(*x*) und cot(*x*) ermöglichen.

(a) Zeichnen Sie zu beiden Klassen ein Klassendiagramm. Machen Sie kenntlich, wie Ihre neuen Klassen mit dem System der Klassen aus der Vorlesung zusammenhängen (Vererbung). Geben Sie die Klassendiagramme als ***.pdf***-Datei ab.

(b) Implementieren Sie beide Klassen in Java. Berechnen Sie den Funktionswert von tan(*x*) und cot(*x*) ausschließlich mit den Klassen aus der Vorlesung unter Verwendung der Beziehungen

$$\tan(x) = \frac{\sin(x)}{\cos(x)} \text{ und } \cot(x) = \frac{\cos(x)}{\sin(x)}.$$

Geben Sie die beiden Dateien ***Tangens.java*** und ***Cotangens.java*** ab.

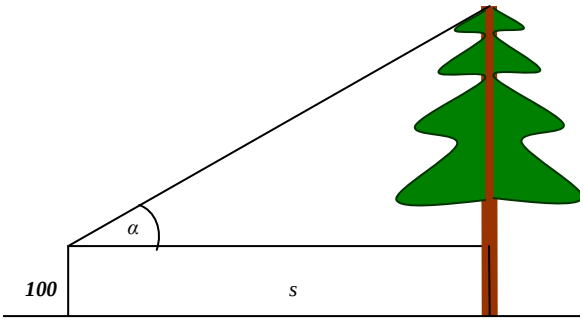
(c) Bekanntlich gilt $\tan(\alpha) = \cot(90° - \alpha)$. Berechnen Sie unter Verwendung Ihrer beiden Klassen `Tangens` und `Cotangens` in einem Programm ***TanCotVergleich.java*** die Funktionswerte $\tan(\alpha)$ und $\cot(90° - \alpha)$, wobei α die Werte 0°, 15°, 30°, 45°, 60°, 75° und 90° durchläuft. Vergleichen Sie die Ergebnisse beider Ausdrücke mit den Ergebnissen der Java-eigenen Methode `Math.tan( x)`. Speichern Sie die Ausgaben mittels *Ausgabenumleitung* in eine Datei ***TanCotVergleich.out***.
Geben Sie die Datei ***TanCotVergleich.out*** zusammen mit dem Programm ***TanCotVergleich.java*** ab.
Hinweis: Beachten Sie, dass Winkelfunktionen stets Argumente im Bogenmaß verlangen.

2. BaumHoehe.java

Eine Försterei untersucht regelmäßig das Wachstum der Bäume ihrer Schonungen. Dazu werden die Bäume vermessen und die Höhe der Bäume unterteilt in 6 Wachstumsgrößen:

Qualitätsgruppe	bis Höhe [cm]	Nutzung
1	50	Setzling
2	150	Unterholz
3	300	Weihnachtbäume
4	450	Brennholz
5	600	Industrieholz
6	höher	Bestandsholz

Mit einem Messinstrument wird von einem festen Standort 100 cm über dem Waldboden der Erhebungswinkel α zwischen der Horizontalen und dem Baumwipfel gemessen. Außerdem wird der Abstand *s* vom Vermessungspunkt bis zum Fußpunkt des Baumes bestimmt. Aus diesen Daten wird die Gesamthöhe des Baumes berechnet.



Schreiben Sie ein Programm ***BaumHoehe.java***, welches mit den Funktionsklassen aus der Vorlesung und aus der 1. Aufgabe Baumhöhen beliebig vieler Bäume berechnet.

(a) Für jeden Baum werden die gemessenen Werte eingegeben. Falls die Eingabewerte verwertbar sind, wird die berechnete Baumhöhe ausgegeben, sonst wird auf die Fehlerhaftigkeit der Messwerte des Baumes hingewiesen. Testen Sie Ihr Programm für die folgenden Messwerte:

Baum 1 (Abstand: 100 cm, Winkel 45°)
Baum 2 (Abstand: 200 cm, Winkel -45°)

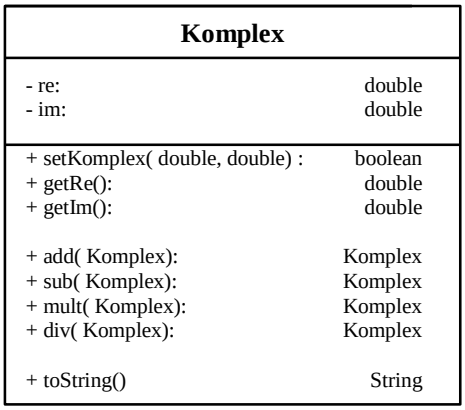
Hinweis: Beachten Sie, dass Winkelfunktionen stets Argumente im Bogenmaß verlangen.

(b) Berechnen Sie die Baumhöhen für die in der Datei ***BaumHoehe.in*** gemessenen Werte:
Für jeden Baum steht in einer Zeile der gemessene Abstand, der gemessene Winkel und, falls ein weiterer Baum folgt, das Zeichen 'j', sonst das Zeichen 'n'. Verwenden Sie für das Einlesen die *Eingabeumlenkung*. Die Ergebnisse schreiben Sie mittels *Ausgabeumlenkung* in eine Datei ***BaumHoehe.out***.

Geben Sie die Datei ***BaumHoehe.out*** zusammen mit dem Programm ***BaumHoehe.java*** ab.

3. ***Komplex.java***

Gegeben sei das folgende Klassendiagramm für eine Klasse `Komplex` der komplexen Zahlen aus *C*, bestehend aus dem Realteil `re`, dem Imaginärteil `im`, den Methoden `add` zur Addition, `sub` zur Subtraktion, `mult` zur Multiplikation und `div` zur Division der komplexen Zahl selbst mit einer anderen.



Ergänzen Sie das Java-Programm ***Komplex.java*** wie folgt:

(a) Nehmen Sie das Attribut für den Realteil `re` und das Attribut für den Imaginärteil `im` auf und geben Sie eine Methode `setKomplex` an, welche der komplexen Zahl den Realteil und den Imaginärteil zuweist.
(b) Schreiben Sie eine Methode `toString` so, dass eine komplexe Zahl in der Form `re + im i` zurückgegeben wird.
(c) Programmieren Sie die Methoden für die Multiplikation `mult` und die Division `div` komplexer Zahlen.
(d) Deklarieren Sie im Testprogramm vier komplexe Zahlen $a,b,c,d \in C$ mit $a = 4 + 2i$ und $b = 3 + 4i$.
(e) Berechnen Sie im Testprogramm die Java-Anweisungen $c = (a+b)*a$ und $d = (a-b)^2$ und geben Sie die Konsolenausgaben für *c* und *d* als Kommentare an.

Hinweis zu Operationen für komplexe Zahlen:

Sei $a,b \in C$ mit $a = a_1 + a_2 i$ und $b = b_1 + b_2 i$, dann gilt

$$a \pm b = (a_1 \pm b_1) + (a_2 \pm b_2)i.$$

$$a * b = (a_1 * b_1) - (a_2 * b_2) + ((a_1 * b_2) + (a_2 * b_1))i$$

$$a / b = \left(\frac{(a_1 * b_1) + (a_2 * b_2)}{b_1^{\,2} + b_2^{\,2}} \right) + \left(\frac{(a_2 * b_1) - (a_1 * b_2)}{b_1^{\,2} + b_2^{\,2}} \right) i$$

Komplex.java (Grobstruktur)

```
/**
 * Komplexe Zahlen
 */
public class Komplex
{
/* ------------------------------------------------------ */
                                                  // (a) Attribute
/**
 * Realteil re einer komplexen Zahl.
 */


/**
 * Imaginaerteil im einer komplexen Zahl.
 */
```

```
/* ------------------------------------------------------ */
                                          // (a) set-Methode
/**
 * Setzt Real- und Imaginaerteil einer komplexen Zahl.
 * @param real Realteil der komplexen Zahl
 * @param imaginaer Imaginaerteil der komplexen Zahl
 * @return true
 */
  public boolean setKomplex( double real, double imaginaer)
  {

  }

/* ------------------------------------------------------ */
                                             // get-Methode
/**
 * Liest Realteil.
 * @return Realteil der komplexen Zahl
 */
  public double getRe()
  {
    return re;
  }

/**
 * Liest Imaginaerteil.
 * @return Imaginaerteil der komplexen Zahl
 */
  public double getIm()
  {
    return im;
  }

/* ------------------------------------------------------ */
                                          // (b) toString
/**
 * Darstellen einer komplexen Zahl.
 * @return komplexen Zahl in linearer Schreibweise
 */
  public String toString()
  {

  }

/* ------------------------------------------------------ */
                                          // (c) Operationen
/**
 * Addition.
 * @param a komplexe Zahl
 * @return komplexe Zahl
 */
  public Komplex add( Komplex a)
  {
    double x, y;

    x = re + a.getRe();
    y = im + a.getIm();

    Komplex b = new Komplex();
    b.setKomplex( x, y);

    return b;
  }
```

```
/**
 * Subtraktion.
 * @param a komplexe Zahl
 * @return komplexe Zahl
 */
  public Komplex sub( Komplex a)
  {
    double x, y;

    x = re - a.getRe();
    y = im - a.getIm();

    Komplex b = new Komplex();
    b.setKomplex( x, y);

    return b;
  }

/**
 * Multiplikation.
 * @param a komplexe Zahl
 * @return komplexe Zahl
 */
  public Komplex mult( Komplex a)
  {


  }

/**
 * Division.
 * @param a komplexe Zahl
 * @return komplexe Zahl
 */
  public Komplex div( Komplex a)
  {


  }

/* ---------------------------------------------------- */
                                        // (d) (e) Anwendung
/**
 * Testprogramm zu komplexe Zahlen
 */
  public static void main( String[] args)
  {
// (d) Komplexe Zahlen a, b, c, d

// (e) Testbeispiel 1

// (e) Testbeispiel 2

// Ausgabe
    System.out.println( "c = " + c);                  // (e)
    System.out.println( "d = " + d);                  // (e)
  }
}
```