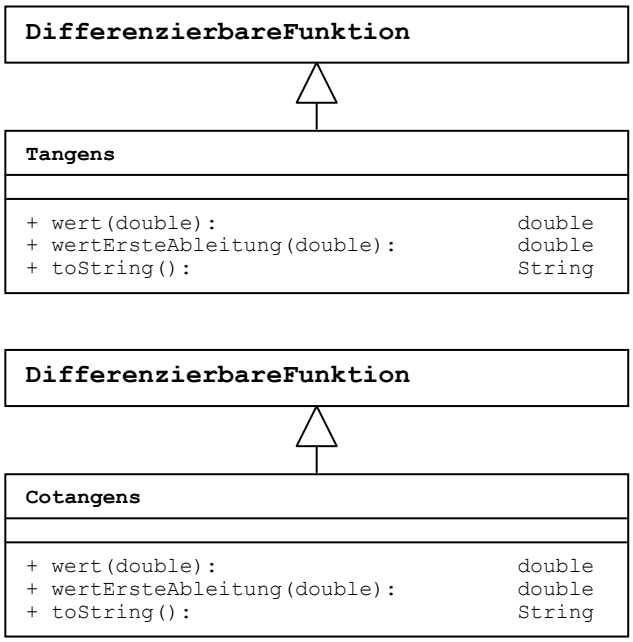
Lösung der Übungsaufgabenserie 12
Grundlagen der Informatik und Numerik

---

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

zu (a)



zu (b)

Tangens.java

```
// Tangens.java                                          MM 2013

/**
 * Tangensfunktion f(x) = tan( x),
 * D = R \ { x | x = PI/2 + n*PI, n aus N }, W = R.
 */
public class Tangens extends DifferenzierbareFunktion
{
/* ------------------------------------------------------ */
                                          // service-Methoden
/**
 * Berechnen eines Funktionswertes durch
 * tan( x) = sin( x)/cos( x).
 * @param arg Argument
 * @return tan( arg)
 */
  public double wert( double arg)
  {
    Sinus sin = new Sinus();
    Cosinus cos = new Cosinus();

    return sin.wert( arg) / cos.wert( arg);
  }

/**
 * Berechnen der ersten Ableitung
 * f'(x) = 1 / (cos( x) * cos( x)).
 * @param arg Argument
 * @return f'( arg)
 */
```

```
    public double wertErsteAbleitung( double arg)
    {
      Cosinus cos = new Cosinus();
      double wert = cos.wert( arg);

      return 1 / (wert * wert);
    }

  /* ------------------------------------------------------ */
                                          // toString-Methode
  /**
   * Darstellen der Tangensfunktion.
   * @return Funktion in linearer Schreibweise
   */
    public String toString()
    {
      return "tan( x)";
    }
  }
```

Cotangens.java

```
  // Cotangens.java                                  MM 2013

  /**
   * Cotangensfunktion f(x) = cot( x),
   * D = R \ { x | x = n*PI, n aus N }, W = R.
   */
  public class Cotangens extends DifferenzierbareFunktion
  {
  /* ------------------------------------------------------ */
                                           // service-Methode
  /**
   * Berechnen eines Funktionswertes durch
   * cot( x) = cos( x) / sin( x).
   * @param arg Argument
   * @return cot( arg)
   */
    public double wert( double arg)
    {
      Sinus sin = new Sinus();
      Cosinus cos = new Cosinus();

      return cos.wert( arg) / sin.wert( arg);
    }

  /**
   * Berechnen der ersten Ableitung
   * f'(x) = - 1 / (sin( x) * sin( x)).
   * @param arg Argument
   * @return f'( arg)
   */
    public double wertErsteAbleitung( double arg)
    {
      Sinus sin = new Sinus();
      double wert = sin.wert( arg);

      return - 1 / (wert * wert);
    }

  /* ------------------------------------------------------ */
                                          // toString-Methode
  /**
   * Darstellen der Cotangensfunktion.
```

```
    * @return Funktion in linearer Schreibweise
    */
     public String toString()
     {
       return "cot( x)";
     }
   }
```

zu (c)

TanCotVergleich.java

```
   // TanCotVergleich.java                                MM 2013

   /**
    * Programm vergleicht Ergebnisse der Klassen
    * Tangens und Cotangens.
    */
   public class TanCotVergleich
   {
   /**
    * tan( x) = cot( PI/2 - x).
    */
     public static void main( String[] args)
     {
   // Neue Funktion
       Tangens tan = new Tangens();
       Cotangens cot = new Cotangens();

   // Wertberechnung
       System.out.println
       ( "Werteberechnung: 0, 15, 30, 45, 60, 75, 90 grad");

       double x = 0, y = Math.PI/2;
       for( int k = 0; k < 7; k++)
       {
         System.out.println();
         System.out.println
         ( "tan( " + x + ")\t= " + tan.wert( x));
         System.out.println
         ( "cot( PI/2 - " + x + ")\t= " + cot.wert( y));
         System.out.println
         ( "Math.tan( " + x + ")\t= " + Math.tan( x));

         x += Math.PI / 12;                        // + 15 grad
         y -= Math.PI / 12;                        // - 15 grad
       }

   // Programm beendet
       System.out.println();
       System.out.println( "Test beendet");
     }
   }
```

TanCotVergleich.out

```
   Werteberechnung: 0, 15, 30, 45, 60, 75, 90 grad

   tan( 0.0)                              = 0.0
   cot( PI/2 - 0.0)                       = 0.0
   Math.tan( 0.0)                         = 0.0

   tan( 0.2617993877991494)               = 0.2679491924311227
   cot( PI/2 - 0.2617993877991494)        = 0.2679491924311224
   Math.tan( 0.2617993877991494)          = 0.2679491924311227
```

```
tan( 0.5235987755982988)              = 0.5773502691896257
cot( PI/2 - 0.5235987755982988)       = 0.5773502691896256
Math.tan( 0.5235987755982988)         = 0.5773502691896257

tan( 0.7853981633974483)              = 1.0
cot( PI/2 - 0.7853981633974483)       = 0.9999999999999999
Math.tan( 0.7853981633974483)         = 0.9999999999999999

tan( 1.0471975511965976)              = 1.7320508075688759
cot( PI/2 - 1.0471975511965976)       = 1.7320508075688756
Math.tan( 1.0471975511965976)         = 1.7320508075688767

tan( 1.308996938995747)               = 3.732050807568875
cot( PI/2 - 1.308996938995747)        = 3.7320508075688723
Math.tan( 1.308996938995747)          = 3.7320508075688745

tan( 1.5707963267948963)              = Infinity
cot( PI/2 - 1.5707963267948963)       = 3.0023997515803315E15
Math.tan( 1.5707963267948963)         = 3.5301143212171575E15

Test beendet
```

2. BaumHoehe.java

BaumHoehe.java

```
// BaumHoehe.java                                    MM 2013
import Tools.IO.*;                                // Eingaben

/**
 * Anwendung der Tangensfunktion zur Berechnung von
 * Baumhoehen.
 */
public class BaumHoehe
{
/**
 * Eingabe der Baumdaten (Abstand, Winkel)
 * Ausgabe der Baumhoehe
 */
  public static void main( String[] args)
  {
    char weiter = 'j';
    do
    {
//Eingabe der Messdaten
      System.out.println();
      double abstand = IOTools.readDouble( "Abstand [cm]: ");
      System.out.println( abstand);

      double winkel = IOTools.readDouble( "Winkel [GRD]: ");
      System.out.println( winkel);

// Berechnung der Baumhoehe
      Tangens tan = new Tangens();
      double hoehe = 100 + abstand
                   * tan.wert( winkel * Math.PI / 180);

// Ausgabe der Baumhoehe
      if( hoehe > 0 && winkel < 90 && winkel > -90)
        System.out.println( "Hoehe: " + hoehe);
      else                        // Fehler in den Messdaten
        System.out.println( "Messfehler in der Eingabe");

// Weiter
```

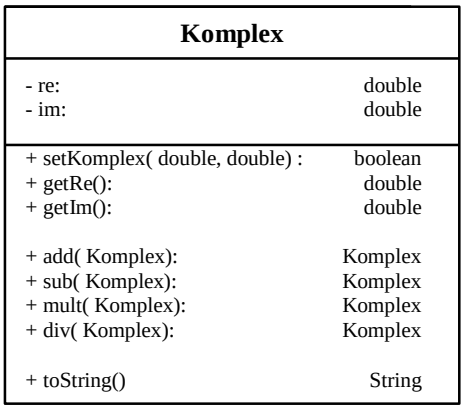
```
        System.out.println();
        weiter = IOTools.readChar( "Noch ein Baum (j/n)? ");
      } while( weiter == 'j');

      System.out.println();
      System.out.println( "Baumberechnung beendet");
    }
  }
```

BaumHoehe.out

```
Abstand [cm]: 84.0
Winkel [GRD]: 80.0
Hoehe: 576.3876728478881

Noch ein Baum (j/n)?
Abstand [cm]: 1120.0
Winkel [GRD]: 45.0
Hoehe: 1220.0

Noch ein Baum (j/n)?
Abstand [cm]: 65.0
Winkel [GRD]: 60.0
Hoehe: 212.58330249197695

Noch ein Baum (j/n)?
Abstand [cm]: 113.0
Winkel [GRD]: -22.0
Hoehe: 54.345036480627286

Noch ein Baum (j/n)?
Abstand [cm]: 135.0
Winkel [GRD]: 16.0
Hoehe: 138.71062707743906

Noch ein Baum (j/n)?
Abstand [cm]: 335.0
Winkel [GRD]: -10.0
Hoehe: 40.930461462664226

Noch ein Baum (j/n)?
Abstand [cm]: 55.0
Winkel [GRD]: -10.0
Hoehe: 90.30201606103442

Noch ein Baum (j/n)?
Abstand [cm]: 20.0
Winkel [GRD]: -45.0
Hoehe: 80.0

Noch ein Baum (j/n)?
Abstand [cm]: 1000.0
Winkel [GRD]: 15.0
Hoehe: 367.9491924311227

Noch ein Baum (j/n)?
Abstand [cm]: 700.0
Winkel [GRD]: 10.0
Hoehe: 223.4288864959255

Noch ein Baum (j/n)?
Abstand [cm]: 90.0
Winkel [GRD]: 80.0
Hoehe: 610.4153637655943
```

```
Noch ein Baum (j/n)?
Abstand [cm]: 100.0
Winkel [GRD]: 45.0
Hoehe: 200.0

Noch ein Baum (j/n)?
Abstand [cm]: 130.0
Winkel [GRD]: 60.0
Hoehe: 325.1666049839539

Noch ein Baum (j/n)?
Abstand [cm]: 135.0
Winkel [GRD]: -22.0
Hoehe: 45.45645951225383

Noch ein Baum (j/n)?
Abstand [cm]: 1135.0
Winkel [GRD]: 35.0
Hoehe: 894.7355558680207

Noch ein Baum (j/n)?
Abstand [cm]: 135.0
Winkel [GRD]: 100.0
Messfehler in der Eingabe

Noch ein Baum (j/n)?
Abstand [cm]: 155.0
Winkel [GRD]: -10.0
Hoehe: 72.66931799018792

Noch ein Baum (j/n)?
Abstand [cm]: 200.0
Winkel [GRD]: -45.0
Messfehler in der Eingabe

Noch ein Baum (j/n)?
Abstand [cm]: 215.0
Winkel [GRD]: 75.0
Hoehe: 902.3909236273096

Noch ein Baum (j/n)?
Abstand [cm]: 250.0
Winkel [GRD]: 90.0
Messfehler in der Eingabe

Noch ein Baum (j/n)?
Baumberechnung beendet
```

3. ***Komplex.java***



Komplex.java

```
// Komplex.java                                          MM 2013

/**
 * Komplexe Zahlen
 */
public class Komplex
{
/* ------------------------------------------------------ */
                                                  // Attribute
/**
 * Realteil einer komplexen Zahl.
 */
  private double re;

/**
 * Imaginaerteil einer komplexen Zahl.
 */
  private double im;

/* ------------------------------------------------------ */
                                                // 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)
  {
    re = real;
    im = imaginaer;

    return true;
  }

/* ------------------------------------------------------ */
                                                // 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;
      }

    /* ------------------------------------------------------ */
                                                        // toString
    /**
     * Darstellen einer komplexen Zahl.
     * @return komplexen Zahl in linearer Schreibweise
     */
      public String toString()
      {
         return "" + re + " + " + im + " i";
      }

    /* ------------------------------------------------------ */
                                                    // 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)
      {
```

```
      double x, y;
      x = re * a.getRe() - im * a.getIm();
      y = re * a.getIm() + im * a.getRe();

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

      return b;
    }

  /**
   * Division.
   * @param a komplexe Zahl
   * @return komplexe Zahl
   */
    public Komplex div( Komplex a)
    {
      double x, y;
      x = ( re * a.getRe() + im * a.getIm())
      / ( a.getRe() * a.getRe() + a.getIm() * a.getIm());
      y = ( im * a.getRe() - re * a.getIm())
      / ( a.getRe() * a.getRe() + a.getIm() * a.getIm());

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

      return b;
    }

  /* ------------------------------------------------- */
  /**
   * Testprogramm zu komplexe Zahlen.
   */
    public static void main( String[] args)
    {
  // Komplexe Zahlen
      Komplex a = new Komplex();
      Komplex b = new Komplex();
      Komplex c = new Komplex();
      Komplex d = new Komplex();
      a.setKomplex( 4, 2);
      b.setKomplex( 3, 4);

  // Testbeispiel 0
  //    c = a.add( b);

  // Testbeispiel 1
      c = a.add( b);
      c = c.mult( a);

  // Testbeispiel 2
      d = a.sub( b);
      d = d.mult( d);

      System.out.println( "c = " + c);   // c = 16.0 + 38.0 i
      System.out.println( "d = " + d);   // d = -3.0 + -4.0 i
    }
  }
```