Aufgabe 13.1 (Ü) Polymorphie und Interfaces

Gegeben seien folgende Java-Klassen:

```
class A {
    public void f() { System.out.println("f() in A"); }
    public void g() { System.out.println("g() in A"); f(); }
}
class B extends A {
    public void f() { System.out.println("f() in B"); }
    public void h() { System.out.println("h() in B"); f(); g(); }
}
class C extends A {
    public void h() { System.out.println("h() in C"); f(); g(); }
}
interface I {
    public void m();
}
abstract class D {
    public abstract void n(boolean b);
    public void o() { System.out.println("o() in D"); n(false); }
}
class E extends D {
    public void n(boolean b) {
        System.out.println("n() in E");
        if(b) o();
    }
    public void m() { System.out.println("m() in E"); }
}
class F extends E implements I {
    public void n(boolean b) {
        System.out.println("n() in F");
    }
}
```

a) Folgende Anweisungen werden vom Compiler erfolgreich übersetzt:

i) `A a = new A();`	☒ Richtig	☐ Falsch
ii) `B b = new B(); A t = b;`	☒ Richtig	☐ Falsch
iii) `B b = new A();`	☐ Richtig	☒ Falsch
iv) `D d = new D();`	☐ Richtig	☒ Falsch
v) `D d = new F();`	☒ Richtig	☐ Falsch
vi) `F f = new F(); I i = f;`	☒ Richtig	☐ Falsch

b) Welche Ausgaben produzieren die folgenden Anweisungen?

i) `A a = new A(); a.g();`

```
g() in A
f() in A
```

ii) `B b = new B(); b.h();`

```
h() in B
f() in B
g() in A
f() in B
```

iii) `C c = new C(); c.g();`

```
g() in A
f() in A
```

iv) `B b = new B(); A a = b; a.g();`

```
g() in A
f() in B
```

v) `D d = new E(); d.n(true);`

```
n() in E
o() in D
n() in E
```

vi) `F f = new F(); f.o();`

```
o() in D
n() in F
```

Aufgabe 13.2 (Ü) Fragestunde

Nutzen Sie die restliche Zeit dazu Ihrem Tutor fragen zu stellen. Bitte bereiten Sie Ihre fragen vor und lassen Sie diese vorher Ihrem Tutor per Email zukommen.

Aufgabe 13.3 (H) Interfaces

In dieser Aufgabe sollen die Klassen aus der Hausaufgabe von Blatt 12 (Geometrische Flächen und Prismen) erweitert werden.

a) Einerseits sollen Klassen vom Typ `Prisma` und `Grundflaeche` untereinander und miteinander vergleichbar gemacht werden in dem beide Klassen das Interface `java.lang.Comparable` implementieren.
   - Prismen werden nach ihrem Volumen (ist das Volumen gleich, dann nach Oberflläche) verglichen;
   - `Grundflaechen` nach ihrer Fläche verglichen;
   - Prismen werden über ihre `Grundflaeche` mit anderen `Grundflaechen` verglichen;

b) Testen sie die Implementierung in dem sie ein Array von `Grundflaeche` mittels `java.util.Arrays.sort()` sortieren.

c) Als zweiten Schritt sollen die Methoden `istQuadrat` und `zuQuadrat` in ein Interface `Quadrierbar` ausgelagert werden, welches nur von solchen `Grundflaechen` implementiert werden soll, die auch ein Quadrat darstellen können.

d) Fügen sie eine weitere `Grundflaeche Dreieck` zur Modellierung von rechtwinkligen Dreiecken zu den bereits bestehenden `Grundflaechen` hinzu. Ein (rechtwinkliges) Dreieck wird durch die Länge der Hypotenuse und Ankathete parametrisiert.

e) Zuletzt erstellen wir ein weiteres Interface `Polygon` mit nur einer Methode `int getEckenAnzahl()`, welche die in einer Grundflläche enthaltenen Ecken zurückgibt.

f) Implementieren sie `Polygon` für alle `Grundflaechen`, die eine endliche Anzahl von Ecken besitzen.

g) Testen sie `Quadrierbar` und `Polygon` in dem sie über ein sortiertes Array vom Typ `java.lang.Comparable`, welches `Prisma` und `Grundflaechen` enthält, iterieren und mittels `instanceof` die Verfügbarkeit beider Interfaces abfragen und, falls vorhanden, die aus diesen Interfaces ermittelbaren Informationen ausgeben.

Lösungsvorschlag 13.3

```
public abstract class Grundflaeche implements java.lang.Comparable {

        public int compareTo(Object o)
        {
                if(o instanceof Grundflaeche)
                        return (int)(flaeche()-((Grundflaeche)o).
                           flaeche());
                if(o instanceof Prisma)
                {
                        if(((Prisma)o).volumen()!=0.0)
                                return -1;
                        else
                                return this.compareTo(((Prisma)o).
                                   getBasis());
                }
                else throw new java.lang.ClassCastException();
        }

    public abstract double umfang();

    public abstract double flaeche();

    @Override
    public String toString() {
        return "␣{Umfang:␣" + umfang() + ";␣Flaeche:␣"+ flaeche() + '}'
            ;
    }
}

public class Kreis extends Grundflaeche{
    private double radius;

    @Override
    public double umfang() {
        return 2*Math.PI*radius;
    }

    @Override
    public double flaeche() {
        return Math.PI*Math.pow(radius,2);
    }

    public Kreis(double radius){
        this.radius = radius;
    }

    public double getRadius(){
        return radius;
    }
```

```
    @Override
    public String toString() {
        return "Kreis{" + "radius=" + radius + super.toString() +'}';
    }


}

public class Rechteck extends Grundflaeche implements Quadrierbar,
   Polygon{
    private double breite;
    private double laenge;

    public double getBreite() {
        return breite;
    }

    public double getLaenge() {
        return laenge;
    }

    @Override
        public int getEckenAnzahl()
        {
                return 4;
        }

        @Override
    public double umfang() {
        return 2*breite+2*laenge;
    }

    @Override
    public double flaeche() {
        return breite*laenge;
    }

    public Rechteck(double breite, double laenge){
        this.breite = breite;
        this.laenge = laenge;
    }

    @Override
     public boolean istQuadrat(){
        return laenge==breite;
    }
    @Override
     public Quadrat zuQuadrat(){
        if (istQuadrat()){
            return new Quadrat(breite);
        } else {
            return null;
        }
     }

    @Override
```

```
    public String toString() {
        return "Rechteck{" + "breite=" + breite + ", laenge=" + laenge
           + super.toString() + ", ecken="+getEckenAnzahl() + '}';
    }



}

public class NEck extends Grundflaeche implements Quadrierbar,Polygon{
    private int n;
    private double laenge;

    public double getLaenge() {
        return laenge;
    }

    public int getN() {
        return n;
    }

    public NEck(int n, double laenge) {
        this.n = n;
        this.laenge = laenge;
    }

    @Override
        public int getEckenAnzahl()
        {
                return n;
        }

    @Override
    public double umfang() {
        return n*laenge;
    }

    @Override
    public double flaeche() {
       return n*Math.pow(laenge,2)/4*Math.tan(Math.PI/n);
    }

    @Override
    public boolean istQuadrat(){
        return n==4;
    }

    @Override
    public Quadrat zuQuadrat(){
        if (istQuadrat()){
            return new Quadrat(laenge);
        } else {
            return null;
        }
    }

    @Override
```

```
    public String toString() {
        return "NEck{" + "n=" + n + ", laenge=" + laenge+ super.
            toString() + ", ecken="+getEckenAnzahl()  + '}';
    }

}

public class Quadrat extends Grundflaeche implements Quadrierbar,
    Polygon{
    private double laenge;

    public double getLaenge() {
        return laenge;
    }

    @Override
        public int getEckenAnzahl()
        {
                return 4;
        }

    @Override
    public double umfang() {
        return 4*laenge;
    }

    @Override
    public double flaeche() {
        return laenge*laenge;
    }

    @Override
    public boolean istQuadrat(){
        return true;
    }

    @Override
    public Quadrat zuQuadrat(){
        return this;
    }

    public Quadrat(double laenge) {
        this.laenge = laenge;
    }

    @Override
    public String toString() {
        return "Quadrat{" + "laenge=" + laenge+ super.toString() + ", "
            ecken="+getEckenAnzahl()  + '}';
    }

}

public class Prisma implements java.lang.Comparable{
    private Grundflaeche basis;
    private double hoehe;
```

```
        public int compareTo(Object o)
        {
                if(o instanceof Prisma)
                    if (Math.abs(volumen()-((Prisma)o).volumen()) <
                       Double.MIN_NORMAL)
                        return (int)(oberflaeche()-((Prisma)o).
                           oberflaeche());
                    else
                        return (int)(volumen()-((Prisma)o).volumen());
                if(o instanceof Grundflaeche)
                {
                        return basis.compareTo(o);
                }
                else throw new java.lang.ClassCastException();
        }

    public Grundflaeche getBasis() {
        return basis;
    }

    public double getHoehe() {
        return hoehe;
    }

    public Prisma(Grundflaeche basis, double hoehe) {
        this.basis = basis;
        this.hoehe = hoehe;
    }

    public double oberflaeche(){
        return hoehe*basis.umfang()+2*basis.flaeche();
    }

    public double volumen(){
        return hoehe*basis.flaeche();
    }

    @Override
    public String toString() {
        return "Prisma{" + "basis= " + basis + ", hoehe= " + hoehe + ",
           Volumen= " +volumen() +
                ", Oberflaeche= " +oberflaeche() + '}';
    }

}

public class Dreieck extends Grundflaeche implements Polygon{
        private double hypotenuse,ankathethe;
        //abstand gibt an wie weit der dritte punkt mit distanz hoehe
           zur grundlinie
        //parallel zum mittelpunkt der grundlinie verschoben ist

    public double getGrundLaenge() {
        return hypotenuse;
    }
    public double geSeitenlaenge() {
        return ankathethe;
```

```
    }

    @Override
        public int getEckenAnzahl()
        {
                return 3;
        }

    @Override
    public double umfang() {
        return hypotenuse + Math.sqrt(hypotenuse*hypotenuse -
            ankathethe*ankathethe);
    }

    @Override
    public double flaeche() {
        return 0.5*Math.sqrt(hypotenuse*hypotenuse - ankathethe*
            ankathethe)*ankathethe;
    }

    public Dreieck(double grundlaenge,double hoehe,double abstand) {
        this.hypotenuse = grundlaenge;
        this.ankathethe = hoehe;
    }

    @Override
    public String toString() {
        return "Dreieck{" + "grundlaenge=" + hypotenuse+ ",␣
            seitenlaenge="+ankathethe + super.toString()  + ",␣ecken="+
            getEckenAnzahl() + '}';
    }

}

public class Test {
   public static void main (String[] args) {

          Grundflaeche gf[]=new Grundflaeche[6];
      gf[0] = new Kreis(7);
      gf[1] = new Rechteck(2,3);
      gf[2] = new Rechteck(4, 4);
      gf[3] = new NEck(7, 3);
      gf[4] = new NEck(4, 5);
      gf[5] = new Dreieck(4, 5, 6);

          System.out.println(java.util.Arrays.toString(gf));
          java.util.Arrays.sort(gf);
          System.out.println(java.util.Arrays.toString(gf));
      System.out.println();

          Prisma pf[]=new Prisma[6];
      pf[0] = new Prisma(gf[0], 0);
      pf[1] = new Prisma(gf[1], 2);
      pf[2] = new Prisma(gf[2], 1);
      pf[3] = new Prisma(gf[3], 3);
      pf[4] = new Prisma(gf[4], 2);
      pf[5] = new Prisma(gf[5], 1);
```

```
        System.out.println(java.util.Arrays.toString(pf));
        java.util.Arrays.sort(pf);
        System.out.println(java.util.Arrays.toString(pf));
    System.out.println();

        java.lang.Comparable cf[]=new java.lang.Comparable[3];
        cf[0]=pf[1];
        cf[1]=gf[1];
        cf[2]=pf[0];

        System.out.println(java.util.Arrays.toString(cf));
        java.util.Arrays.sort(cf);
        System.out.println(java.util.Arrays.toString(cf));

        System.out.println();
        for(Grundflaeche i : gf)
        {
                if(i instanceof Quadrierbar)
                {
                        System.out.println(i+" ist Quadriebar");
                        if(((Quadrierbar)i).istQuadrat())
                                System.out.println(((Quadrierbar)i).
                                    zuQuadrat()+"=="+i);
                        System.out.println();
                }
                else
                        System.out.println(i+" ist NICHT Quadriebar")
                            ;
        }

    }
}
```