Übungsaufgaben 4

4.1 Vererbung und Polymorphie

a) Ausgehend von der bekannten Klassenhierarchie aus Abb. 1 implementieren Sie in den Klassen Figur, Rechteck und Quadrat einen **eigenen parameterlosen Konstruktor**.
Überschreiben Sie in den Klassen Figur, Rechteck und Quadrat die `toString()`-Methode. Vermeiden Sie redundanten Code durch entsprechenden Aufruf von `super` in der `toString()`-Methode der jeweiligen Klasse (d.h. nutzen Sie die Vererbungshierarchie aus). Erzeugen Sie in der Testklasse ein Quadrat-Objekt `q`, ein Rechteck-Objekt `r` und ein Figur-Objekt `f`. Welche Ausgaben erwarten Sie bei folgenden Aufrufen in der Testklasse:

```
(1) System.out.println(f);
(2) System.out.println(r);
(3) System.out.println(q);
```

b) Definieren Sie in den Klassen Figur, Rechteck und Quadrat eine `public`-**Instanzvariable** `farbe` vom Typ `String` und initialisieren Sie es unterschiedlich in der jeweiligen Klasse (z.B. mit gelb, rot und blau). Implementieren Sie *nur* in der Klasse Figur eine Getter-Methode dazu, nämlich `getFarbe()`. Welche Ausgaben erwarten Sie bei folgenden Aufrufen in der Testklasse:

```
(4) System.out.println(f.farbe);
(5) System.out.println(r.farbe);
(6) System.out.println(q.farbe);

(7) System.out.println(f.getFarbe());
(8) System.out.println(r.getFarbe());
(9) System.out.println(q.getFarbe());
```

c) Überschreiben Sie in der Klasse Quadrat die Methode `getFarbe()`. Was wird jetzt in der Testklasse bei den Aufrufen (7) – (9) ausgegeben?

d) Integrieren Sie nun die Instanzvariable `farbe` in die `toString()`-Methode der jeweiligen Klasse, so dass bei der Ausgabe eines Objektes vom Typ Figur, Quadrat und Rechteck ebenfalls seine Farbe ausgegeben wird. Wiederholen Sie die Aufrufe (1) – (3). Was wird ausgegeben?

e) Überschreiben Sie nun auch in den Klassen Quadrat und Rechteck die Methode `getFarbe()`. Die Implementierung der `toString()`-Methode in der Klasse Figur soll jetzt wie folgt aussehen:

```
public String toString() {
      return "Anker: " + anker + " Farbe: " + getFarbe();
}
```

Die `toString()`-Methode der Klasse Quadrat korrigieren Sie jetzt wie folgt (und die Implementierung von `toString()` in der Klasse Rechteck analog dazu, d.h. ohne explizite Ausgabe von `farbe`):

```
public String toString() {
      return super.toString() + " breite = " + breite;
}
```

Was wird jetzt in der Testklasse bei den Aufrufen (1) – (3) ausgegeben?

f) Implementieren Sie jetzt in der Klasse Rechteck eine Instanzmethode `printFarben()`, welche die Farbe vom Rechteck, seinem Vater Quadrat und seinem Großvater Figur auf dem Bildschirm ausgibt. Testen Sie die Methode durch einen entsprechenden Aufruf in der Testklasse.

g) Definieren Sie in der Klasse Figur, Rechteck und Quadrat eine `public`-**Klassenvariable** `var` vom Typ `float` und initialisieren Sie diese unterschiedlich in der jeweiligen Klasse (z.B. mit 1.1f, 2.2f und 3.3f). In der Testklasse geben Sie nun den Wert der Klassenvariable `var` aus der Klasse Rechteck, Quadrat bzw. Figur auf dem Bildschirm aus.

h) Berechnen Sie die Fläche von dem Rechteck-Objekt mithilfe der (eigenen) Rechteck-**Instanzmethode** `berechneFlaeche()`. Berechnen Sie dann die Fläche von diesem Rechteck mithilfe der Quadrat-**Instanzmethode** `berechneFlaeche()`.

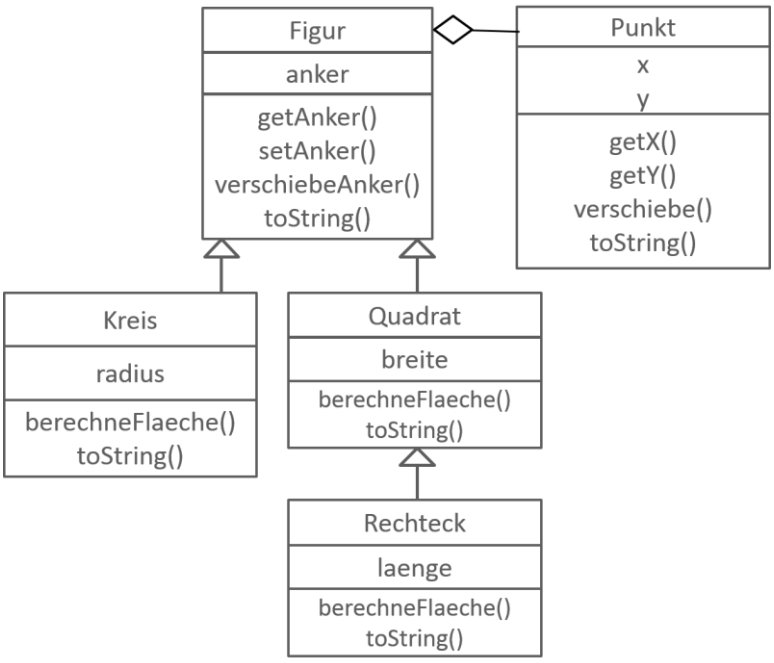

Abb. 1: Klassenhierarchie

4.2 Klassenhierarchie korrigieren

Verändern Sie die Klassenhierarchie aus der Abb. 1 dahingehend, dass die Klasse Rechteck von der Klasse Figur erbt und die Klasse Quadrat vom Rechteck erbt. Damit hätten Sie die in der Mathematik übliche Beziehung: ein Rechteck **ist eine** Figur, ein Quadrat **ist ein** Rechteck.

Implementieren Sie in den Klassen Rechteck und Quadrat die nötigen Methoden, so dass man die Fläche und den Umfang eines Rechtecks und eines Quadrats mithilfe dieser Methoden berechnen kann.

Diskutieren Sie, welche Klassenhierarchie besser (effizienter, übersichtlicher, verständlicher) ist?