Übung 07: Vererbung

Aufgabe 1: Datentypen

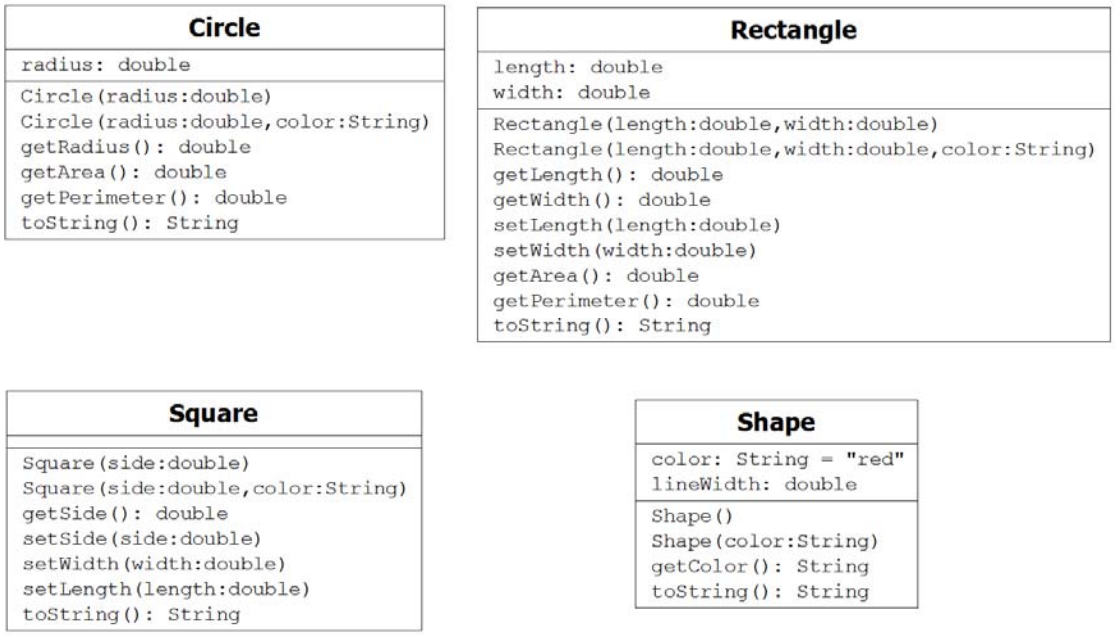
Gegeben sei eine Oberklasse *O*. Eine Unterklasse *U* erbt von der Oberklasse *O*. Welche der nachfolgenden Ausdrücke a) bis f) sind im Anschluss an die beiden ersten Codezeilen erlaubt, welche nicht? Begründen Sie Ihre Antwort!

```
Oberklasse o = new Oberklasse();
Unterklasse u = new Unterklasse();
   a) o = u;
   b) u = o;
   c) u = new Oberklasse();
   d) o = new Unterklasse();
   e) Oberklasse o1 = u; Unterklasse u1 = o1;
   f) Unterklasse u1 = u; Oberklasse o1 = u1;
```

Können Sie mit Hilfe eines Typecasts eines der Statements korrigieren, so dass der Compiler keinen Alarm schlägt?

Aufgabe 2: Geometrische Formen

In der folgenden Aufgabe sollen Vererbungskonzepte in Java an einem kleinen Beispiel erprobt werden. Die folgende Abbildung zeigt die zu implementierenden Klassen.



Im Klassendiagramm sind noch keine Sichtbarkeiten eingetragen, ebenso fehlen die Beziehungen (Assoziation, Vererbung) zwischen den Klassen.

a) Überlegen Sie sich, wie sie die 4 Klassen in einer Vererbungshierarchie anordnen können! Fertigen Sie ggfs. eine Zeichnung an.

b) Implementieren Sie alle 4 Klassen unter Beachtung der UML Klassendiagramme und der folgenden Vorgaben ***auf der nächsten Seite***:

- Die **`toString()`**-Methode gibt jeweils die typischen Merkmale einer Klasse als String zurück. *Beispiel*: Für eine Rechteck könnte z.B. zurückgegeben werden: `Rectangle(width=1.0, length=2.0) is a Shape(color=blue, line_width=0.82)` zurückgegeben werden. Bei einer abgeleiteten Klasse sollen also auch die Merkmale der Oberklasse und deren Typ zurückgegeben werden. Lösen Sie das elegant!
- Die Linienstärke (**`lineWidth`**) einer Form (`Shape`) ist eine Zufallszahl aus dem Intervall [0; 2[.
- Schränken Sie ***Sichtbarkeiten*** (`private`, `public`, etc.) soweit als möglich ein.
- Vermeiden Sie ***Code-Redundanz*** soweit als möglich.
- Alle Klassen müssen ***immutable*** sein mit 2 Ausnahmen:
  - Man darf von jeder Klasse eine Unterklasse ableiten.
  - Bei den Klassen `Rectangle` und `Square` sind „Setter“ erlaubt.
- Überschreiben Sie ggfs. Funktionen in den Unterklassen, z.B. `setWidth()` und `setLength()` in `Square`. Verwenden Sie die Annotation `@Override`.
- Konstanten und mathematische Funktionen finden Sie in der Klasse **`Math`**: https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

c) Schauen Sie sich die Vererbungshierarchie in IntelliJ durch Ctrl + h an.

d) Implementieren Sie in einer JUnit4 Klasse die folgenden 3 „Tests“:

- Die Fläche eines Kreises mit Radius 2 beträgt ca. 12,6. Hinweis: http://junit.sourceforge.net/junit3.8.1/javadoc/junit/framework/Assert.html#assertEquals(double,%20double,%20double)

- Ändert man mit der Methode `setLength()` die Länge eines Quadrats (`Square`), so ist danach das Attribut `length` der Oberklasse gleich dem Attribut `width` der Oberklasse.

- Bauen Sie ein Array aus 4 Elementen mit einem Kreis, einem Quadrat, einem Rechteck und einer allgemeinen geometrischen Form. Iterieren Sie dann über das Arrray und geben Sie dabei die Merkmale aller Objekte auf der Konsole aus. Warum funktioniert das?