Aufgabe 1.1
Im Download-Bereich zu dieser Übung finden Sie den Quellcode einer Klasse `Person`.

a) Schreiben Sie eine Unterklasse `Stud`, die die Klasse `Person` erweitert.
Als zusätzliches privates Attribut haben Stud-Objekte eine ganzzahlige Matrikelnummer, die durch den Aufruf des Konstruktors fortlaufend automatisch generiert wird (beginnend bei 100). Auch für dieses Attribut soll es eine get-Methode `matNr()` geben.
Die `toString()`-Methode wird modifiziert: statt des Geburtsjahres soll die Matrikelnummer und der Name (in dieser Reihenfolge) geliefert werden.

b) Schreiben Sie eine **abstrakte** Klasse `Sportler`, die ebenfalls von `Person` abgeleitet wird.
Sportler haben ein zusätzliches `int`-Attribut `groesse` (in cm) und ein zusätzliches `double`-Attribut `gewicht` (in kg). Werte für beide Attribute müssen dem Konstruktor übergeben werden. Stellen Sie get-Methoden für beide Attribute zur Verfügung.
Sportler verfügen außerdem über eine (abstrakte) Methode `info()`, die einen String mit zusätzlichen Informationen über das Objekt liefert.
`toString()` gibt für Sportler nicht nur den Namen und das Geburtsjahr, sondern auch diese zusätzliche Information zurück.

c) Implementieren Sie eine (konkrete) Klasse `Boxer`, die von `Sportler` abgeleitet ist.
Boxer verfügen außerdem über eine Methode `gewichtsklasse`, die abhängig vom Gewicht einen der Strings „Leicht“ (bis 70 kg), „Mittel“ (zwischen 70 und 90 kg) oder „Schwer“ (ab 90 kg) liefert.
Das Gewicht eines Boxers kann sich ändern, hierfür soll es eine set-Methode `setGewicht(double)` geben. (Achtung! Zugrifffsrechte)
Die `info()`-Methode für Boxer soll die Gewichtsklasse des Boxers liefern.

d) Implementieren Sie eine weitere (konkrete) Unterklasse von Sportlern, nämlich `Fussballer`.
Fussballer haben ein `String`-Attribut `verein` und ein boolesches Attribut `linksfuss`, dass angibt, ob der Spieler rechts- oder links-füßig spielt.
Die `info())`-Methode für Fussballer gibt den Verein des Spielers an.

e) Überschreiben Sie in den beiden Klassen `Person` und `Stud` auch die Methode `equals()`:
Personen sind gleich, wenn sie gleich heißen und das gleiche Geburtsjahr haben.
Studierende sind gleich, wenn sie die gleiche Matrikelnummer haben.
Sportler werden wie Personen verglichen.

Benutzen Sie die *Annotation* `@Override`, um sicher zu stellen, dass Sie tatsächlich die entsprechende Methode der Klasse `Object` überschreiben! Diese erwartet und akzeptiert Objekte der „Mutter aller Klassen“ `Object`! Sie müssen also zunächst prüfen, ob das übergebene Objekt einen passenden Typ hat, bevor Sie den entsprechenden expliziten downcast anwenden können. Verwenden Sie dafür die `getClass()`-Methode wie in folgenden Code-Snippet:

```
if (!( this . getClass () . equals (o. getClass ()))) return false ;
```

Machen Sie sich noch einmal den Unterschied zwischen „==“ und „ `equals` “ klar!

f) Im Material zu dieser Übung finden Sie eine Klasse mit JUnit-Tests, mit deren Hilfe Sie Ihre Implementierungen testen können.
Beachten Sie insbesondere das Verhalten von `equals()` in Bezug auf diejenigen Eigenschaften, die in der Java-API unter `java.lang.object.equals` beschrieben sind. ($\rightsquigarrow$ *Äquivalenzrelation*)
(Die Bemerkungen zu `hashCode()` können Sie ignorieren.)

g) Schreiben Sie dann (und **erst dann**, wenn alle JUnit-Tests „bestanden“ wurden!) eine main-Klasse mit einer main-Methode, in der Sie einige Objekte der Klassen erzeugen und sich mit print-Anweisungen auf dem Monitor anzeigen lassen.

Aufgabe 1.2
Betrachten Sie das untenstehende Klassendiagramm und implementieren Sie es. Die Klasse Punkt2D modelliert Punkte im $\mathbb{R}^2$. Eine Implementierung dieser Klasse finden Sie im Download-Bereich zur Übung. Verwenden Sie für notwendige Rechnungeng aus der Java-API: Math.PI, Math.abs(), Math.sqrt().
Ergänzen Sie die JUnit-Tests.

- Ein Kreis ist durch seinen Mittelpunkt und den Radius gegeben, ein Quadrat durch den Eckpunkt links unten und die Kantenlänge. Die Koordinaten des rechten oberen Eckpunkts können daraus berechnet werden.
- Die Angaben zu Radius bzw Kantenlänge in den Konstruktoren der Klassen Kreis bzw Quadrat dürfen zwar negativ sein, aber bei negativen Eingaben soll immer mit dem Betrag des Wertes gearbeitet werden. Entsprechendes gilt für den Skalierungsfaktor der Methode `multiply()`.
- Die Klasse Vektor2D modelliert Vektoren im $\mathbb{R}^2$. Vektoren unterscheiden sich von Punkten (nur) dadurch, dass man mit Vektoren rechnen kann. Definieren Sie für die Klasse Vektor2D vier Konstruktoren:
  einen mit den zwei double-Komponenten als Eingabeparameter;
  einen zweiten mit einem Punkt als Eingabeparameter („Ortsvektor“ zu dem Punkt),
  einen dritten mit zwei (End-)Punkten als Eingabeparameter („Richtungsvektor“)
  und eine vierten parameterlosen (Standard-)Konstruktor, der den Nullvektor erzeugt.
- Bei der Methode add() sind Eingabeparameter und Returnwert jeweils vom Typ Addierbar. Sie müssen bei der Implementierung in der Klasse Vektor2D allerdings auf Methoden der Klasse Vektor2D bzw Punkt2D zurückgreifen, die für Objekte des Typs Addierbar nicht zwingend bekannt sind. Nehmen Sie daher einen expliziten cast des Eingabeparameters auf den Typ Vektor2D vor.