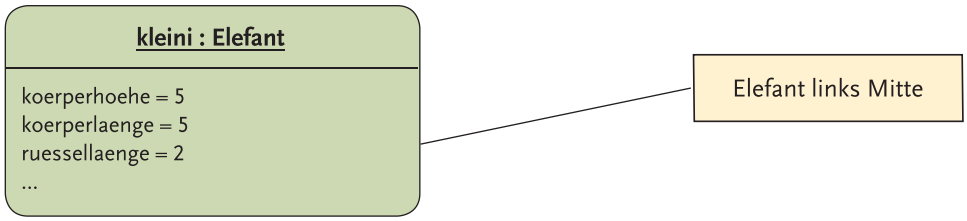
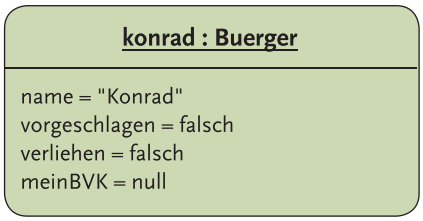


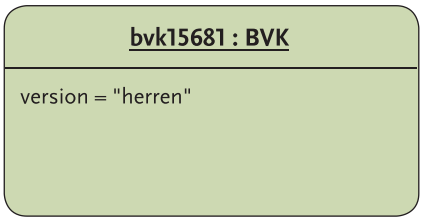
c) In der zweiten Abbildung wird ersichtlich, dass der Elefant Elmar, verglichen mit seinen Artgenossen aus dem ersten Bild, ungewöhnlich bunt ist. Dies würde man evtl. in einem Attribut wie *muster* speichern. In allen anderen Attributen unterscheidet er sich jedoch nicht von den grauen Elefanten. Was also beim Betrachten des Bildes sofort ins Auge sticht, erscheint bei einer Auflistung von mindestens zehn Attributen nicht so ungewöhnlich.

SB S. 123

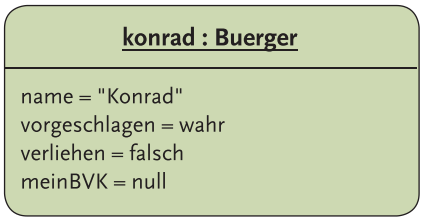
Aufgabe 2 Objekte

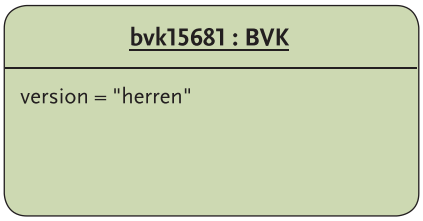
a) i)



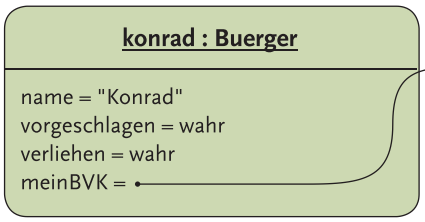


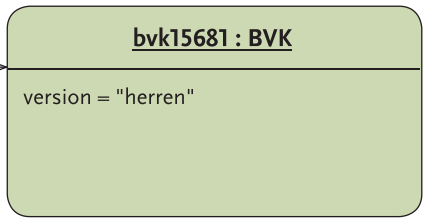
ii)





iii)





b) In einem Entwurfsdiagramm ist *name* und *version* vom Typ *Text*. Der Name muss auch tatsächlich als String implementiert werden, während bei *version* auch ein *char* ausreichen würde. Die Attribute *vorgeschlagen* und *verliehen* sind Wahrheitswerte, da es für diese Attribute jeweils nur zwei mögliche Werte gibt. Evtl. könnte man hier aber auch ein Datum als Zahl eintragen. Das Attribut *meinBVK* ist vom Datentyp *BVK* und verweist hier bei i) und ii) auf null und nach der Verleihung auf bvk15681.

c) Ein Objekt wird als Instanz einer Klasse erzeugt. Dabei fungiert die Klasse als Bauplan und legt über Attribute die Eigenschaften aller Objekte dieser Klasse fest. Die Attribute können je nach Objekt aber (in aller Regel) mit unterschiedlichen Werten belegt sein. Die in der Klasse programmierten Methoden geben die Fertigkeiten der Objekte vor.
Von den Objekten aus gesehen ist die Klasse deren abstrakte Verallgemeinerung in der Form eines Bauplans. So lassen sich mehrere gleichartige Objekte in einer Klasse zusammenfassen.

SB S. 124

Aufgabe 3 Klassendiagramme

a) Das Entwurfsdiagramm beinhaltet zwei Klassen, *Braut* und *Braeutigam*, die jeweils mit einer Assoziation miteinander verbunden sind. Die Assoziationen heißen *ehemann* bzw. *ehefrau* und haben die Multiplizität 1. Die beiden Klassen sind in ihrem Aufbau identisch und besitzen die Attribute *vorname*, *nachname*, *persoNr*, *verheiratet*, *hochzeitstag* und *ring*. Den Attributen ist jeweils einer der drei möglichen Datentypen

(Zahl, Text, Wahrheitswert) zugewiesen. Darunter stehen die Methoden der Klasse *jawortGeben* und *ringAnstecken*.

b) Beide Modellierungen stellen ein Ehepaar dar, einmal mit zwei Klassen und einmal mit einer Klasse modelliert. Die zweite Modellierung trägt dem Umstand Rechnung, dass die Klassen aus der ersten identisch sind. Da beide Klassen mit der anderen über eine Assoziation verbunden sind, besitzt die Klasse der zweiten Modellierung eine Assoziation zu sich selbst.
Die Vorteile der ersten Modellierung ist evtl. ihre Übersichtlichkeit, auch wenn dann im Prinzip eine doppelte Codierung vorliegt. Dieses sollte, wenn möglich, vermieden werden. Die zweite Modellierung ist in dieser Hinsicht sparsamer, aber eben nicht so übersichtlich. Auf jeden Fall ist die zweite Modellierung in der Lage, auch gleichgeschlechtliche Partnerschaften besser darzustellen.

c) Bei der Überführung müssen die „einfachen" Datentypen der Attribute in entsprechende programmiersprachenspezifische Typen umgewandelt werden. Es muss ein Konstruktor festgelegt werden, ebenso wie get- und set-Methoden für die Attribute. Die Methoden des Entwurfsdiagramms müssen mit Parametern und ihren Typen sowie mit Rückgabetypen für Anfragen ausgestattet werden. Assoziationen können übernommen werden, es muss jedoch die Sichtbarkeit (*public* oder *private*) wie bei allen Attributen und Methoden vermerkt werden.

d)

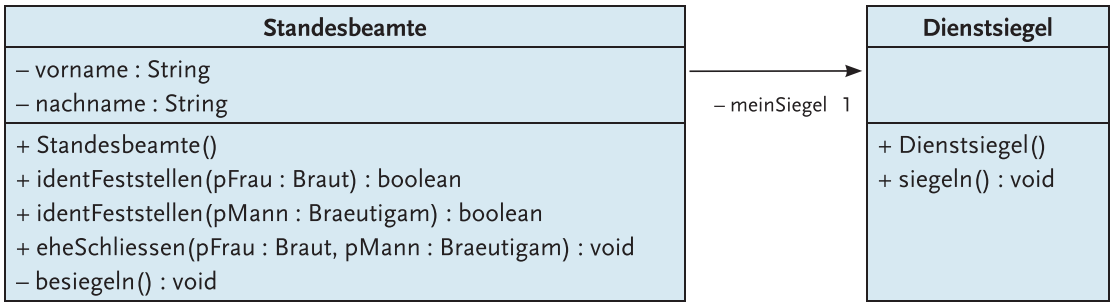
Braut
– vorname : String – nachname : String – persoNr : int – verheiratet : boolean – ring: boolean – hochzeitstag : int
+ Braut(pVor : String, pNac : String, persoNr : int, pHoc : int) + getVorname() : String + getNachname() : String + getPersoNr() : int + getVerheiratet() : boolean + setVerheiratet(verheiratet : boolean) : vold + getRing() : boolean + setRing(pRing : boolean) + getHochzeitstag() : int + setHochzeitstag(pHochzeitstag : int) : vold + jaWortGeben(pOK : boolean) + ringAnstecken() + setNachname(Nachname : String) + getEhemann() : Braeutigam

e) i)
```
public Braut(String pVor, String pNac, int persoNr, int pHoc)
  {
    this.persoNr = persoNr;
    this.vorname = pVor;
    this.nachname = pNac;
    this.verheiratet = false;
    this.ring = false;
    this.hochzeitstag = pHoc;
  }
```

ii)
```
public boolean getVerheiratet()
  {
    return verheiratet;
  }
```

```
public void setVerheiratet(boolean pVerheiratet) {
  this.verheiratet = pVerheiratet;
}
```

f)



e) Der Standesbeamte kann den Namen eines Ehepartners ändern, dieses wird am Beispiel der Braut erklärt. Der Beamte kann über die Methode *getEhemann()* auf den Bräutigam und somit auch seinen Nachnamen zugreifen. Der Zugriff auf den Nachnamen erfolgt ebenso über eine get-Methode des Bräutigams. In der Methode *nameAendern* des Standesbeamten wird dann über den Parameter *pFrau* deren Methode *setNachname* aufgerufen und als Parameter dieser Methode wird der Nachname des Bräutigams übergeben.

SB S. 125

Aufgabe 4 Objektkommunikation

a) Der Kunde *leo* möchte Geld von einer Währung in die andere wechseln. Dies wird durch die Methode *geldWechseln* angestoßen, die die Summe der Ausgangswährung und die gewünschte Tauschwährung als Parameter erhält. Die Bankerin *kati* übernimmt die beiden Parameter für ihre Methode *umtauschen* und lässt wiederum mithilfe der beiden Parameter *tab*, ein Objekt der Klasse *Kursrechner* den korrelierenden Betrag in der neuen Währung ausrechnen. Dieser wird als double-Wert zurückgegeben. Von dem so ermittelten Betrag wird durch einen internen Methodenaufruf vom Bankerobjekt *kati* eine Gebühr abgezogen, bevor der Wert an *leo* zurückgemeldet wird.

b)
```
public double betragErrechnen(int summe, String waehrung) {
double betrag = 0.0;
if(waehrung.equals("Dollar"))
{
    betrag = summe*1,38;
}
else
{
    betrag=summe*0,83;
}
return betrag;
}
```

SB S. 127

Aufgabe 5 Vererbung

a) Bei der Vererbung werden mindestens zwei Klassen in eine hierarchische Beziehung gesetzt. Es gibt eine Oberklasse und eine oder mehrere Unterklassen, die wiederum Unterklassen besitzen können. Betrachtet man die Vererbungsstruktur von der Oberklasse zu den Unterklassen, spricht man davon, dass die Unterklassen die Oberklasse spezialisieren. Betrachtet man die Vererbungsstruktur von den Unterklassen zur Oberklasse, so bedeutet dies, dass die Oberklasse ihre gemeinsamen Unterklassen generalisiert. Das besondere an der Vererbung ist, dass alle Unterklassen (auch über mehrere Ebenen hinweg) die Attribute und Methoden der Oberklasse besitzen, ohne dass diese erneut modelliert oder implementiert werden müssen.