Aufgabe 2

a) Um eine Klasse zu serialisieren muss sie das Interface *Serializable* implementieren oder Unterklasse einer Klasse sein, die serialisierbar ist.

d) Weder die Klassen der Simple-Properties noch eine ihrer Oberklassen implementieren das Interface *Serializable*, was nach a) eine Voraussetzung für die Serialisierung eines Objektes ist.

f) Wird eine serialisierbare Klasse verändert und neu erzeugt, ändert sich auch die serialVersionUID. Falls sich die serialVersionUID des Objektes nun beim deserialisieren von der aktuellen serialVersionUID der Klasse unterscheidet, führt dies zu einer *InvalidClassException.*

g) Die serialVersionUID ist ein Hashcode aus Eigenschafen der Klasse, wie Namen, Attributen, Parametern, Sichtbarkeit und so weiter. Sie wird als *long* wie ein Attribut gespeichert. Ändert sich der Aufbau einer Klasse, ändert sich der Hashcode und damit auch die serialVersionUID.

h) Durch Serialisierung kann man tiefe Kopien von Objekten erzeugen, die im Gegensatz zur Zuweisung an eine andere Variable oder der clone()-Methode eines Objektes nicht die Referenz an das Objekt kopiert, sondern das eigentliche Objekt. Dazu muss die Instanz einer Klasse serialisiert werden, anschließend deserialisiert und zu einem Objekt konvertiert werden.

Aufgabe 3

c) Die XML-Serialisierung verwendet statt eines ObjectOutputStreams (binäre Serialisierung) oder eines XStream-Objekts (XStream) einen XMLEncoder und statt eines ObjectInput-Streams (binäre Serialisierung) oder eines XStream-Objekts (XStream) einen XMLDecoder. Klassen, die durch XML-Serialisierung serialisiert werden sollen, müssen nicht das Interface *Serializable* implementieren.
Bei der Serialisierung mit Java Beans muss eine bestimmte Designkonvention erfüllt werden und jedes Attribut muss mit einem Getter und einem Setter erreichbar gemacht werden. Zudem wird ein Standardkonstruktor benötigt.

d) Binäre Serialisierung ist weitaus effizienter als Serialisierung über XML. Das binäre Format ist kompakter und spart damit Speicherplatz und/oder Bandbreite. Das Format wird schneller gelesen und geschrieben.
Nachteil ist die schwierige Weiterverarbeitung von Nicht-Java-Programmen oder die nachträgliche Änderung ohne Einlesen und Wiederaufbauen der Objekte.
XML ist dort flexibler. Das Format ist nicht Java-spezifisch und auch das manuelle Editieren ist ohne Aufwand möglich. Werte lassen sich so direkt auslesen oder anpassen, allerdings hängt es vom Anwendungsfall ab, ob dies als Vorteil gesehen wird, denn auch private Attribute werden serialisiert und können gelesen und verändert werden.