Hochschule Ludwigshafen am Rhein

Klausur WS 2014/2015

Bitte arbeiten Sie elektronisch Lösungen zu den folgenden Aufgaben auf dem zugewiesenen PC im Prüfungsraum aus. Erstellen Sie für Ihren Workspace einen Ordner „workspace“ auf dem Netzlaufwerk „P:/“.

Packen Sie Ihre Lösungen am Ende der Klausur in eine ZIP-Datei und nennen Sie diese Sxxxxxx-Klausur.zip (xxxxxx = Matr.Nr.). Speichern Sie die ZIP-Datei auf dem Laufwerk „P:/“, lassen Sie Eclipse geöffent und den PC auch nach Ende der Klausur angeschaltet.

Achten Sie auf gut formatierten Code.

Aufgaben 1 -3 sind zu Anfang anzufertigen. Unter Umständen bietet Ihnen Eclipse an, diese in einem Rutsch zu lösen. Die anderen Aufgaben bauen darauf auf.

1 Erstellen der benötigten Klassen (8 Punkte)

a) Entwickeln Sie das Projekt `KoenigDerLoewen` im Package `de.hs_lu.ws1415.klausur`. Setzen Sie mit möglichst wenig Aufwand folgende Klassen um:

`Tier`	
bezeichnung name zahm	Attribut vom Typ `String` Attribut vom Typ `String` Attribut vom Typ `Boolean`

`Geier`	
aasfresser	Attribut vom Typ `Boolean`

`Saeugetier`	

`Loewe`	
isKoenig	Attribut vom Typ `Boolean`

`Giraffe`	
halslaenge	Attribut vom Typ `Double`

b) Implementieren Sie explizit die parameterlosen Default-Konstruktoren in ihren Klassen.
c) Statten Sie die Klassen mit geeigneten Getter- und Settermethoden aus.

2 Umsetzen einer geeigneten Datenstruktur und weitere Konstruktoren (8 Punkte)

a) Sorgen Sie dafür, dass

- `Loewe` und `Giraffe` von `Saeugetier`
- `Saeugetier` von `Tier`

alle Methoden und Attribute erben. Erweitern Sie Ihre Klassen mit dem/den entsprechenden Schlüsselwort, bzw. Schlüsselwörtern.

b) Statten Sie die Klassen mit weiteren geeigneten Konstruktoren aus. Sorgen Sie dafür, dass bei Benutzung der neuen Konstruktoren **alle** Variablen gefüllt werden, auch die der Superklasse.

3 Implementieren Sie eine `main-Methode` in der Klasse `VerwaltungApp` (14 Punkte)

a) Erstellen Sie eine Klasse `TierweltApp` und statten Sie diese während der Erzeugung oder manuell mit einer `main-Methode` aus.

b) Erzeugen Sie in der `main-Methode` folgende Objekte:

- Ein `Loewe` mit Variablenname „`simba`“ und
  - Bezeichnung: „`Loewe`“
  - Name: „`Simba`“
  - Zahm: `false`
  - isKoenig: `true`
- Ein `Geier` mit Variablenname „`flinn`“ und
  - Bezeichnung: „`Geier`“
  - Name: „`flinn`“
  - Zahm: `true`
  - Aasfresser: `true`
- Eine `Giraffe` mit Variablenname „`marius`“ und
  - Bezeichnung: „`Giraffe`“
  - Name: „`marius`“
  - Zahm: `true`
  - Halslaenge: `2.40`

c) Aufgrund neuer Forschungsergebnisse wurde festgestellt, dass Geier doch auch lebende Tier angreifen und deshalb nicht zahm sind. Ändern Sie das nachträglich und setzen sie beim Geier die Variable „`zahm`“ mit der entsprechenden Methode auf `false`.

d) Sorgen Sie dafür, dass keine Objekte vom Typ `Tier` als auch vom Typ `Saeugetier` erstellt werden können. Kommentieren Sie notfalls nicht funktionsfähigen Code wieder aus!

e) Verpflichten Sie alle erbenden Klassen von `Tier` eine `toString-Methode` zu implementieren. Beheben Sie die Ursache der Complilerwarnungen. Achten Sie dabei auf eine gut formatierte Ausgabe und nutzen sie in jeder Ausgabe mindestens 2 besondere Maskierungen für Ausgaben von Sonderzeichen.

f) Erstellen Sie in der Klasse `Tier` eine Methode `isTheSameAnimal (Object otherObject)`, welche einen boolean zurückgibt und zeigen Sie, dass es sich bei `simba` und `flinn` nicht um die gleichen Objekte handelt.

g) Erklären Sie kurz in einem Kommentar im Quellcode zur Methode der vorherigen Aufgabe `isTheSameAnimal` welche Möglichkeiten es zum Objektvergleich gibt und welche man in in welchen Fällen anwendet, damit diese über alle JVMs hinweg das gleiche Ergebnis liefern.

4 Exceptions (8 Punkte)

a) Programmieren Sie eine Exception mit Klassenname `TrinktKeineMilchException`, welche im aufgerufenen Fall die Meldung mit der Meldung „Trinkt keine Milch, da es sich um ein Saeugetier handelt. Bitte beachten!“ ausgibt.

Tipp: Zusätzliche Variablen und Methoden werden nicht zwingend benötigt.

b) Erstellen Sie in der Klasse `TierweltApp` eine Methode `pruefeLaktoseToleranz(Tier tier)`. Prüfen Sie in der Methode, ob es sich um ein `Saeugetier` handelt. Geben Sie im postitiven Falle true zurück. Falls das Tier kein `Saeugetier` ist, ***werfen*** Sie eine `TrinktKeineMilchException`, behandeln Sie diese in der aufrufenden Methode und geben Sie diese auf die Konsole aus.

c) Prüfen Sie, ob `flinn` laktosetolerant ist. Tipp: Denken Sie objektorientiert. Sollte der Aufruf nicht objektorientiert funktionieren, können Sie Teilpunkte mit der statischen Vorgehenweise bekommen.

5 Schnittstelle `KannFliegen` (10 Punkte)

a) Erstellen Sie ein Interface `KannFliegen`. Sorgen Sie dafür, dass dieses Interface von `Geier` implementiert wird.

b) Statten Sie dieses Interface mit einer Methode `getFlugFormation()` aus. Beheben Sie die Komplierwarnungen und geben Sie im Bedarfsfall einfach einen String mit Inhalt „V-Formation“ zurück.

c) Erstellen Sie in der `TierweltApp` einen Kommentar und **begründen** Sie, wo ein Interface `Fortbewegung` mit der Methode `getFortbewegungsart()` sinnvollerweise implementiert werden sollte.

6 Erstellen eines Sortiments mit Collections (12 Punkte)

a) Erstellen Sie eine Klasse `BiologischeOrdnung`, die von ArrayList erbt.

b) Erstellen Sie eine Klasse `Artenvielfalt`, die

- eine Variable `saeugetiere` vom Typ `BiologischeOrdnung`, die nur Saeugetiere aufnimmt und
- eine Variable `fleischfresser` vom Typ `BiologischeOrdnung`, die nur `Geier` und `Loewen` aufnimmt. Dafür gibt es 2 Möglichkeiten. Sie können z.b. wie in Aufgabenblatt 8 Nr. 2 c vorgehen. So Sie weitere Klassen oder Interfaces benötigen, zögern Sie nicht diese zu erstellen.

c) Implementieren Sie einen Konstruktor und die dazugehörigen Getter & Setter

d) Erzeugen Sie in der `main-Methode` der Klasse `TierweltApp` eine Variable mit Namen `artenvielfalt` vom Typ `Artenvielfalt`. Fügen Sie folgende Objekte in folgende Collection hinzu:

- `saeugetiere`: `simba & marius`
- `fleischfresser`: `simba & flinn`

Zeigen Sie, dass `fleischfresser` keine anderen Objekte zulässt. Fügen Sie dafür einfach `marius` hinzu. Falls Sie einen Fehler vom Compiler bekommen, kommentieren Sie den Code einfach wieder aus, lassen ihn aber ansonsten unverändert stehen!

e) Nutzen Sie eine `foreach-Schleife, Enumeration` oder einen `Iterator` und geben Sie nur `Saeugetiere` aus `fleischfresser` auf die Konsole aus. Prüfen Sie dazu jedes Objekt ab, bevor sie es auf die Konsole ausgeben.

Bemerkung: die `toString` gibt nur einen leeren String zurück. Es erscheint keinerlei Ausgabe auf der Konsole. Das ist kein Grund zur Beunruhigung. Wenn Sie Zeit übrig haben, können Sie die `toString` nachbessern.

7 Zusatzaufgabe Cloneable (6 Zusatzpunkte)

a) Sorgen Sie dafür, dass **alle** `Tiere` das Interface `Cloneable` implementieren.
b) Begründen Sie im Kommentar, warum die Methode `clone()` abstrakt sein muss und imlementieren diese in mindestens einer beliebigen Klasse aus.
c) Sorgen Sie dafür, dass Ihre `Artenvielfalt` die Schnittstelle `Cloneable` implementiert. Stellen sie sicher, dass Ihr Code eine DeepCopy, auch tiefe Kopie genannt, erstellt.

Bemerkung: Beachten Sie, dass Fleischfresser auch Tiere sind. Tiere wiederum sind `Cloneable`.

Viel Erfolg!

Hochschule Ludwigshafen am Rhein

Programmieren II - Klausur SS2015

Bitte arbeiten Sie elektronisch Lösungen zu den folgenden Aufgaben auf dem zugewiesenen PC im Prüfungsraum aus. Erstellen Sie für Ihren Workspace einen Ordner „workspace“ auf dem Netzlaufwerk „P:/“.

Packen Sie Ihre Lösungen am Ende der Klausur in eine ZIP-Datei und nennen Sie diese Sxxxxxx-Klausur.zip (xxxxxx = Matr.Nr.). Speichern Sie die ZIP-Datei auf dem Laufwerk „P:/“, lassen Sie Eclipse geöffent und den PC auch nach Ende der Klausur angeschaltet.

Achten Sie auf gut formatierten Code.

Aufgaben 1 -3 sind zu Anfang anzufertigen. Unter Umständen bietet Ihnen Eclipse an, diese in einem Rutsch zu lösen. Die anderen Aufgaben bauen darauf auf.

1 Erstellen der benötigten Klassen (8 Punkte)

a) Entwickeln Sie das Projekt `Europapolitik` im Package `de.hs_lu.ss2015.klausur`. Setzen Sie mit möglichst wenig Aufwand folgende Klassen um:

`Waehrung`	
bezeichnung	Attribut vom Typ `String`

`Staat`	
bezeichnung	Attribut vom Typ `String`
euStaat	Attribut vom Typ `Boolean`
schengenStaat	Attribut vom Typ `Boolean`
bruttoInlandsProdukt	Attribut vom Typ `int`
waehrung	Attribut vom Typ `Waehrung`

`Geberstaat`	

`Pleitestaat`	

`Drittstaat`	

b) Implementieren Sie explizit die parameterlosen Default-Konstruktoren in ihren Klassen.
c) Statten Sie die Klassen mit geeigneten Getter- und Settermethoden aus.

2 Umsetzen einer geeigneten Datenstruktur und weitere Konstruktoren (8 Punkte)

a) Sorgen Sie dafür, dass

- `Geberstaat`, `Pleitestaat` und `Drittstaat` von `Staat`

alle Methoden und Attribute erben. Erweitern Sie Ihre Klassen mit dem/den entsprechenden Schlüsselwort, bzw. Schlüsselwörtern.

b) Statten Sie die Klassen mit weiteren geeigneten Konstruktoren aus. Sorgen Sie dafür, dass bei Benutzung der neuen Konstruktoren **alle** Variablen gefüllt werden, auch die der Superklasse.

3 Implementieren Sie eine `main-Methode` in der Klasse `PolitikApp` (14 Punkte)

a) Erstellen Sie eine Klasse `PolitikApp` und statten Sie diese während der Erzeugung oder manuell mit einer `main-Methode` aus.

b) Erzeugen Sie in der `main-Methode` folgende Objekte:

- Eine `Waehrung` mit Variablenname „`euro`“ und
  - Bezeichnung: „`EURO`“
- Eine `Waehrung` mit Variablenname „`goldmuenzen`“ und
  - Bezeichnung: „`Goldmünzen`“
- Eine `Waehrung` mit Variablenname „`franken`“ und
  - Bezeichnung: „`Franken`“
- Einen `Pleitestaat` mit Variablenname „`griechenland`“ und
  - Bezeichnung: „`Griechenland`“
  - EU-Staat: `true`
  - Schengen-Staat: `true`
  - Bruttoinlandsprodukt: 19247
  - Währung: `Variable mit Namen „euro“`
- Einen `Geberstaat` mit Variablenname „`deutschland`“ und
  - Bezeichnung: „`Deutschland`“
  - EU-Staat: `true`
  - Schengen-Staat: `true`
  - Bruttoinlandsprodukt: 34219
  - Währung: `Variable mit Namen „euro“`
- Einen `Drittstaat` mit Variablenname „`schweiz`“ und
  - Bezeichnung: „`Schweiz`“
  - EU-Staat: `false`
  - Schengen-Staat: `true`
  - Bruttoinlandsprodukt: 71037
  - Währung: `Variable mit Namen „franken“`

c) Aufgrund politscher Entscheidungen, steigt Griechenland aus dem Euro aus. Ändern Sie die Währung nachträglich und setzen Sie die Variable mit der entsprechenden Methode auf „`goldmuenzen`“.

d) Alle Staaten, die keine `Pleitestaaten` oder `Geberstaaten` sind, sind automatisch `Drittstaaten`. Sorgen Sie dafür, dass keine Objekte vom Typ `Staat` erstellt werden können. Kommentieren Sie notfalls nicht funktionsfähigen Code wieder aus!

e) Verpflichten Sie alle erbenden Klassen von `Staat` eine `toString-Methode` zu implementieren. Beheben Sie die Ursache der Complilerwarnungen. Achten Sie dabei auf eine gut formatierte Ausgabe und nutzen **sie in einer Ausgabe mindestens 2 besondere Maskierungen** für Ausgaben von Sonderzeichen.

f) Erstellen Sie in der Klasse `Waehrung` eine Methode `isTheSameCurrency (Object otherObject)`, welche einen boolean zurückgibt und geben Sie auf der Konsole (bitte den Aufruf an geeigneter Stelle implementieren!!) aus, dass es sich bei den Währungen von `deutschland` und der `schweiz` nicht um die gleichen Objekte handelt.

g) Erklären Sie kurz in einem Kommentar im Quellcode zur Methode der vorherigen Aufgabe `isTheSameCurrency` welche Möglichkeiten es zum Objektvergleich gibt und welche man in welchen Fällen anwendet, damit diese über alle JVMs hinweg das voraussagbare bzw. beeinflussbare Ergebnisse liefern.

4 Exceptions (10 Punkte)

a) Programmieren Sie eine Exception mit Name `KeinenEuroOhneMitgliedschaftException`, welche im aufgerufenen Fall die Meldung mit der Meldung „Ohne EU-Mitgliedschaft keinen EURO!“ ausgibt.

Tipp: Zusätzliche Variablen und Methoden werden nicht zwingend benötigt. Warnungen bzgl. Serial Version ID können ignoriert werden!

b) Erstellen Sie in der Klasse `PolitikApp` eine Methode `moechteEuroBeitreten(Staat staat)`. Prüfen Sie in der Methode, ob es sich um einen Staat handelt, bei dem `euStaat` den Wert `true` hat. Geben Sie im positiven Falle einen String mit Wert “`OK`“ zurück. Andernfalls ***werfen*** Sie eine `KeinenEuroOhneMitgliedschaftException`, behandeln Sie diese in der aufrufenden Methode und geben Sie diese auf die Konsole aus.

c) Prüfen Sie, ob die Schweiz dem Euro beitreten kann. Tipp: Denken Sie objektorientiert. Sollte der Aufruf nicht objektorientiert funktionieren, können Sie Teilpunkte mit der statischen Vorgehensweise bekommen.

5 Schnittstelle `EURecht` (8 Punkte)

a) Erstellen Sie ein Interface `EURecht`. Sorgen Sie dafür, dass dieses Interface von `Pleitestaat` und `Pleitestaat` implementiert wird.

b) Statten Sie dieses Interface mit einer Methode `getEuRecht()` aus, dessen Rückgabetyp `String` ist. Beheben Sie die Komplierwarnungen und geben Sie im Bedarfsfall einfach `null` oder einen leeren String zurück.

c) Erstellen Sie in der `PolitikApp` einen Kommentar und **begründen** Sie, wo ein Interface `Recht` mit der Methode `getNationaleGesetze()` sinnvollerweise implementiert werden sollte.

6 Erstellen eines Sortiments mit Collections (12 Punkte)

a) Erstellen Sie eine beliebige Collection mit dem Variablennamen `natostaaten`. Sorgen Sie dafür, dass zu dieser Collection nur Staaten zugelassen werden.

b) Fügen Sie die Variablen `deutschland` und `griechenland` hinzu.

c) Beweisen Sie, dass die Variable `euro` nicht hinzugefügt werden kann. Kommentieren Sie, falls nötig, nicht funktionierenden Code wieder aus

d) Nutzen Sie eine `foreach-Schleife`, `Enumeration` oder einen `Iterator` und geben Sie nur `Geberstaat` aus `natostaaten` auf die Konsole aus. Prüfen Sie dazu jedes Objekt ab, bevor sie es auf die Konsole ausgeben.

e) Angenommen die Klasse `Staat` hätte eine ausprogrammierte Methode `toString()`, von welche `toString()` würde dann aufgerufen werden und warum? Begründen Sie Ihre Antwort in einem Kommentar in der Klasse `PolitikApp`.

7 Zusatzaufgabe Comparable (6 Zusatzpunkte)

a) Sorgen Sie dafür, dass **alle Staaten** das Interface `Comparable` implementieren.
b) Fügen Sie alle Staaten einer ArrayList mit Namen `staaten` hinzu in der `main-Methode` und geben Sie diese auf die Konsole aus.
c) Sortieren sie die Staaten nach dem Bruttoinlandsprodukt und geben sie die Staaten erneut auf die Konsole aus.

Viel Erfolg!

Probeklausur

Bitte arbeiten Sie elektronisch Lösungen zu den folgenden Aufgaben auf dem zugewiesenen PC im Pruefungsraum aus. Packen Sie Ihre Lösungen in eine ZIP-Datei und benennen Sie diese Sxxxxxx-Klausur.zip (xxxxxx = Matr.Nr.). Speichern Sie die Zip-Datei auf den Desktop und lassen Sie den PC auch nach Ende der Klausur angeschaltet. Achten Sie auf gut formatierten Code.

1. Erstellen der benoetigten Klassen

a) Erstellen Sie einen Workspace an beliebiger Stelle. Entwickeln Sie im Projekt `Fahrzeugverwaltung` das Package `de.hs_lu.ws2013.klausur` um setzen Sie mit möglichst wenig Aufwand folgende Klassen um:

Motor	
`hubraum`	Attribut vom Datentyp `int`
`leistung`	Attribut vom Datentyp `double`
`kraftstoff`	Attribut vom Datentyp `String`
`nockenwellensteuerung`	Attribut vom Datentyp `String`

Fahrzeug	
`motor`	Attribut vom Typ Motor
`hoechstgeschwindigkeit`	Attribut vom Datentyp `double`
`zulaessigeGesamtmasseInTonnen`	Attribut vom Datentyp `double`
`sitzplaetze`	Attribut von geeignetem Datentyp

Motorrad	
`kategorie`	Attribut vom Datentyp `String`

Bus	
`stehplaetze`	Attribut von geeignetem Datentyp

LKW	
`nutzlast`	Attribut vom Datentyp `double`

Auto	
`karosserieform`	Attribut vom Datentyp `String`

b) Statten Sie die Klassen mit geeigneten Getter- und Settermethoden aus.

c) Implementieren Sie explizit die Default-Konstruktoren in ihren Klassen.

2. Umsetzen einer geeigneten Datenstruktur und weitere Konstruktoren

a) Sorgen Sie dafuer, dass `Motorrad, Bus, LKW und Auto` von `Fahrzeug` alle Methoden und Attribute erben. Erweitern Sie Ihre Klassen mit dem/den entsprechenden Schluesselwort, bzw. Schluesselwoertern.

b) Statten Sie die Klassen mit weiteren geeigneten Konstruktoren aus.Sorgen Sie dafuer, dass bei Benutzung der neuen Konstruktoren **alle** Variablen gefuellt werden, auch die der Superklasse.

3. Implementieren einer `Main-Methode` und in der Klasse `FahrzeugApp`

a) Erstellen Sie eine Klasse `FahrzeugApp` und statten Sie diese waehrend der Erzeugung oder manuel mit einer `Main-Methode` aus.

b) Erzeugen Sie in der `Main-Methode`

folgende Motoren:

- Variablenname „k4m“ und
  - Hubraum: 1598
  - Leistung: 110
  - Kraftstoff: Benzin
  - Nockenwellensteuerung: Zahnriemen
- Variablenname „om601“ und
  - Hubraum: 1997
  - Leistung: 72
  - Kraftstoff: Benzin
  - Nockenwellensteuerung: Steuerkette
- Variablenname „om542“ und
  - Hubraum: 14997
  - Leistung: 653
  - Kraftstoff: Diesel
  - Nockenwellensteuerung: Steuerkette
- Variablenname „jx21“ und
  - Hubraum: 998
  - Leistung: 175
  - Kraftstoff: Benzin
  - Nockenwellensteuerung: Stirnrad

folgende Fahrzeuge und passen Sie falls noetig das Format der Eingabewerte an:

- Auto mit Variablenname „renault“ und
  - Karosserieform: Kombi
  - Motor: k4m
  - Hoechstgeschwindigkeit: 190
  - Zulaessige Gesamtmasse in Tonnen: 1.590
  - Sitzplaetze: 5

- Auto mit Variablenname „mercedes“ und
  - Karosserieform: Limousine
  - Motor: om601
  - Hoechstgeschwindigkeit: 170
  - Zulaessige Gesamtmasse in Tonnen: 2.430
  - Sitzplaetze: 5

- LKW mit Variablenname „arctros“ und
  - Nutzlast: 20
  - Motor: om542
  - Hoechstgeschwindigkeit: 90
  - Zulaessige Gesamtmasse in Tonnen: 19.995
  - Sitzplaetze: 2

- Bus mit Variablenname „citaro“ und
  - Stehplaetze: 48
  - Motor: om542
  - Hoechstgeschwindigkeit: 105
  - Zulaessige Gesamtmasse in Tonnen: 12.590
  - Sitzplaetze: 50

- Motorrad mit Variablenname „ninja“ und
  - Kategorie: Supersport
  - Motor: jx21
  - Hoechstgeschwindigkeit: 390
  - Zulaessige Gesamtmasse in Tonnen: 310
  - Sitzplaetze: 2

c) Verpflichten Sie alle erbenden Klassen von `Fahrzeug` eine `toString-Methode` zu implementieren. Sorgen Sie dafuer, dass keine Objekte vom Typ Fahrzeug erstellt werden koennen. Kommentieren Sie notfalls nicht funktionsfaehigen Code wieder aus.

d) Zeigen Sie, dass im `arctros` und im `citaro` der gleiche Motor verbaut ist. Fragen Sie dazu den `arctros` und packen das in die Variable im `arctrosMotor`. Das selbe noch einmal mit dem `citaro` und packen das in die Variable `citaroMotor`. Speichern Sie das boolsche Ergebnis in die Variable `gleicherMotor`.

Kommentieren Sie desweiteren im Quellcode, ob sie die Objekte vergleichen oder den tatsaechlichen Inhalt der Variablen. Verdeutlichen Sie kurz den unterschied zwischen `==` und .equals.

e) Wegen dem Rasseln der Steuerkette erhaelt der `citaro` einen neuen Motor. Erstellen Sie dazu in der `Main-Methode` eine neue Variable mit dem Namen `om543`. Es aendert sich nur die Nockenwellensteuerung hin zu einem „Zahnriemen“.

4. Exceptions

Erstellen Sie eine Methode `pruefePassstrassentauglichkeit()` mit dem Return-Type `boolean`. Speichern Sie das Ergebnis der Abfragen in jeweile eine eigene Variable. Fuer jedes Fahrzeug, bei dem die zulaessige Gesamtmasse ueber 3,5 Tonnen liegt, soll eine `FahrzeugZuSchwer`**`Exception`** mit der Meldung „Fahrzeug fuer Passstrassen ungeeignet“ geworfen werden. Fangen Sie etwagige Exceptions in der `Main-Methode` ab.

5. Schnittstelle `Nutzfahrzeug`

Erstellen Sie ein Interface `Nutzfahrzeug` und statten Sie dieses mit einer Methode `getReturnOfInvestment()` aus, die einen `double` zurueckgibt. Sorgen Sie dafuer, dass LKW und Bus dieses Interface implementieren aber programmieren Sie die Methode falls möglich nur 1x aus. Geben Sie einen beliebigen `double`-Wert zurück.

6. Erstellen eines Fuhrparks mit Collections

a) Erstellen Sie eine Klasse `Fuhrpark`, die alle Objekte in einer Variable `objects` speichert. Nutzen Sie eine beliebige Collection.

b) erstellen Sie eine Variable `fahrzeuge` die nur Objekte vom typ `fahrzeuge` zulaesst.

c) implementieren Sie die Getter- und Settermethoden und ergaenzen Sie einen Konstruktor.

d) Erzeugen Sie in der `Main-Methode` der Klasse `FahrzeugApp` ein Object vom Typ `Fuhrpark`. Fuegen Sie alle bisher erstellten Objekte in `objects` ein und wiederholen Sie den Vorgang, indem Sie alle Fahrzeuge in die Variable `fahrzeuge` hinzufuegen.

e) Nutzen Sie eine `foreach-Schleife, Enumeration` oder einen `Interator` und geben Sie den kompletten Inhalt der Variable `fahrzeuge` auf die Konsole aus.

7. Comparable

a) Erstellen Sie in der `Main-Methode` der Klasse `FahrzeugApp` eine Variable `fuhrparkSortiertNachLeistung`. Fuegen Sie alle bisher erstellten Objekte aus `fuhrpark` in ein sortiertes Collection-Objekt `fuhrparkSortiertNachLeistung` ein (aehnlich wie Nr. 6 e, diesmal z.B. eine ArrayList). Sorgen Sie wieder dafuer, dass nur Fahrzeuge hinzugefuegt werden koennen. Programmieren Sie diese Methode nur 1x aus.

b) Sorgen Sie dafuer, dass **alle** Fahrzeuge `Comparable` sind. Sortieren Sie die Fahrzeuge nach Leistung.

Fachhochschule
Ludwigshafen am Rhein

Hochschule
für Wirtschaft

Klausur WS2013/14

Bitte arbeiten Sie elektronisch Lösungen zu den folgenden Aufgaben auf dem zugewiesenen PC im Prüfungsraum aus. Erstellen Sie für Ihren Workspace einen Ordner „workspace“ auf dem Netzlaufwerk „P:/“.

Packen Sie Ihre Lösungen am Ende der Klausur in eine ZIP-Datei und nennen Sie diese Sxxxxxx-Klausur.zip (xxxxxx = Matr.Nr.). Speichern Sie die Zip-Datei auf dem Laufwerk „P:/“, lassen Sie Eclipse geöffent und den PC auch nach Ende der Klausur angeschaltet. Achten Sie auf gut formatierten Code.

Aufgaben 1 -3 sind zu Anfang anzufertigen. Unter Umständen bietet Ihnen Eclipse an, diese in einem Rutsch zu lösen. Die anderen Aufgaben bauen darauf auf.

1. Erstellen der benötigten Klassen (8 Punkte)

a) Entwickeln Sie das Projekt `Supermarkt` im Package `de.hs_lu.ws2013.klausur`. Setzen Sie mit möglichst wenig Aufwand folgende Klassen um:

`Artikel`	
`artikelNummer`	Attribut vom Datentyp `int`
`Preis`	Attribut vom Datentyp `double`
`Verkaufsbezeichnung`	Attribut vom Datentyp `String`

`NonFood`	
`garantieBestimmungen`	Attribut vom Typ String

`Food`	
`Kalorien`	Attribut von geeignetem Datentyp

b) Statten Sie die Klassen mit geeigneten Getter- und Settermethoden aus.

c) Implementieren Sie explizit die parameterlosen Default-Konstruktoren in ihren Klassen.

2. Umsetzen einer geeigneten Datenstruktur und weitere Konstruktoren (8 Punkte)

a) Sorgen Sie dafür, dass `Food` und `NonFood` von `Artikel` alle Methoden und Attribute erben. Erweitern Sie Ihre Klassen mit dem/den entsprechenden Schlüsselwort, bzw. Schlüsselwörtern.

b) Statten Sie die Klassen mit weiteren geeigneten Konstruktoren aus. Sorgen Sie dafür, dass bei Benutzung der neuen Konstruktoren **alle** Variablen gefüllt werden, auch die der Superklasse.

3. Implementieren einer `main-Methode` in der Klasse `VerwaltungApp` (14 Punkte)

a) Erstellen Sie eine Klasse `VerwaltungApp` und statten Sie diese während der Erzeugung oder manuell mit einer `main-Methode` aus.

b) Erzeugen Sie in der `main-Methode` folgende Objekte:

- Ein `Food` mit Variablenname „pizza“ und
  - Artikelnummer: 1
  - Preis: 1.99
  - Verkaufsbezeichnung: „Tiefkühlpizza“
  - Kalorien: 858

- Ein `NonFood` mit Variablenname „fahrrad“ und
  - Artikelnummer: 2
  - Preis: 599.00
  - Verkaufsbezeichnung: „Fahrrad“
  - Garantiebestimmungen: „2 Jahre Garantie“

c) Wegen besserer Rohstoffe verändert sich die Verkaufbezeichnung von `fahrrad` in „Fahrrad rostfrei“. Desweiteren gibt es jetzt „5 Jahre Garantie“.

d) Sorgen Sie dafür, dass keine Objekte vom Typ `Artikel` erstellt werden können. Kommentieren Sie notfalls nicht funktionsfähigen Code wieder aus!

e) Verpflichten Sie alle erbenden Klassen von `Artikel` eine `toString-Methode` zu implementieren. Beheben Sie die Ursache der Complilerwarnungen in Food und NonFood. Sie können Zeit sparen, in dem Sie im Methodenrumpf nur ““ zurückgeben.

f) Erstellen Sie in der Klasse `Artikel` eine Methode `isTheSame(Object otherObject)`, welche einen boolean zurückgibt und zeigen Sie, dass es sich bei `pizza` und `fahrrad` nicht um die gleichen Objekte handelt. Es reicht, wenn Sie nur 1 Möglichkeit implementieren. Nennen Sie im Kommentar mindestens 1 weitere Möglichkeit, machen Sie die fehlersichere deutlich.

4. Exceptions (8 Punkte)

a) Programmieren Sie eine Exception mit Klassenname `KeinDiaetFoodException`, welche im aufgerufenen Fall die Meldung mit der Meldung „Zum Abnehmen ungeeignet!“ ausgibt.

Tipp: Zusätzliche Variablen und Methoden werden nicht zwingend benötigt.

b) Erstellen Sie in der Klasse `VerwaltungApp` eine Methode `pruefeDiaetTauglichkeit(Food food)`. Prüfen Sie in der Methode den Kalorienwert ab, und geben Sie im Falle, dass der Kalorienwert 500 übersteigt eine `KeinDiaetFoodException` auf die Konsole aus. Sie können bei Bedarf folgendes Code-Fragment verwenden.:

```
... pruefeDiaetTauglichkeit(Food food){
      double kalorien = food.getKalorien();
      …
}
```

c) Es gibt die Möglichkeit `Exceptions` zu werfen (`throws`). Beschreiben Sie, wo der Fehler in diesem Fall behandelt werden müsste, oder nennen Sie den Namen der entsprechenden Methode. Schreiben Sie das in einen Kommentar in der Methode `pruefeDiaetTauglichkeit(Food food)`.

5. Schnittstelle `UmsatzsteuerReduziert` (10 Punkte)

a) Erstellen Sie ein Interface `UmsatzsteuerReduziert` und statten Sie dieses mit einer Methode `isUmsatzsteuerReduziert()` aus, die einen `boolean` zurückgibt. Sorgen Sie dafür, dass `Food` dieses Interface implementiert. Als Rückgabewert der Methode können Sie einfach den Wert `true` nehmen.

b) Erstellen Sie in der `VerwaltungApp` einen Kommentar und **begründen** Sie, wo ein Interface `UmsatzsteuerPflichtig` mit der Methode `getUmsatzsteuersatz()` sinnvollerweise implementiert werden sollte. Beachten Sie, dass es auf die Artikel unterschiedliche Umsatzsteuersätz geben kann.

6. Erstellen eines Sortiments mit Collections (12 Punkte)

a) Erstellen Sie eine Klasse `Sortiment`, die alle beliebigen Objekte in einer Collection mit Namen `artikel` speichert. Nutzen Sie eine geeignete Collection, die alle „Object“ aufnimmt.

b) Erstellen Sie in der Klasse `Sortiment` eine Collection mit Variablenname `nahrungsmittel` die nur Objekte vom Typ `Food` zulässt.

c) Implementieren Sie in der Klasse `Sortiment` die Getter- und Settermethoden und ergänzen Sie einen Konstruktor.

d) Erzeugen Sie in der `main-Methode` der Klasse `VerwaltungApp` eine Variable mit Namen `sortiment` vom Typ `Sortiment`. Fügen Sie die bisher erstellten Objekte `pizza` und `fahrrad` im `sortiment` in der Variable `artikel` ein.

e) Fügen Sie in der `main-Methode` das Object `pizza` zu `nahrungsmittel` im `sortiment` hinzu.

e) Nutzen Sie eine `foreach-Schleife, Enumeration` oder einen `Iterator` und geben Sie nur `NonFood` aus dem `sortiment` auf die Konsole aus. Sie können dazu, wenn Sie das `sortiment` haben, auf `artikel` zugreifen und überprüfen, ob es sich um `NonFood` handelt.

7. Zusatzaufgabe Comparable (6 Zusatzpunkte)

a) Sorgen Sie dafür, dass **alle** `Artikel` das Interface `Comparable` implemtieren. Sortieren Sie die Artikel nach `Food` oder `NonFood`.

b) Erstellen Sie in der `main-Methode` der Klasse `VerwaltungApp` eine geeignete Collection mit Namen `sortimentSortiert`. Sie können beispielsweise eine ArrayList nutzen. Fügen Sie alle bisher erstellten Objekte aus `sortiment` ein. Sorgen Sie wieder dafür, dass nur `Artikel` hinzugefügt werden können. Sortieren Sie ihre Collection.

Probeklausur

Bitte arbeiten Sie elektronisch Lösungen zu den folgenden Aufgaben auf dem zugewiesenen PC im Pruefungsraum aus. Packen Sie Ihre Lösungen in eine ZIP-Datei und benennen Sie diese Sxxxxxx-Klausur.zip (xxxxxx = Matr.Nr.). Speichern Sie die Zip-Datei auf den Desktop und lassen Sie den PC auch nach Ende der Klausur angeschaltet. Achten Sie auf gut formatierten Code.

1. Erstellen der benoetigten Klassen

a) Erstellen Sie einen Workspace an beliebiger Stelle. Entwickeln Sie im Projekt `Fahrzeugverwaltung` das Package `de.hs_lu.ws2013.klausur` um setzen Sie mit möglichst wenig Aufwand folgende Klassen um:

Motor	
`hubraum`	Attribut vom Datentyp `int`
`leistung`	Attribut vom Datentyp `double`
`kraftstoff`	Attribut vom Datentyp `String`
`nockenwellensteuerung`	Attribut vom Datentyp `String`

Fahrzeug	
`motor`	Attribut vom Typ Motor
`hoechstgeschwindigkeit`	Attribut vom Datentyp `double`
`zulaessigeGesamtmasseInTonnen`	Attribut vom Datentyp `double`
`sitzplaetze`	Attribut von geeignetem Datentyp

Motorrad	
`kategorie`	Attribut vom Datentyp `String`

Bus	
`stehplaetze`	Attribut von geeignetem Datentyp

LKW	
`nutzlast`	Attribut vom Datentyp `double`

Auto	
`karosserieform`	Attribut vom Datentyp `String`

b) Statten Sie die Klassen mit geeigneten Getter- und Settermethoden aus.

c) Implementieren Sie explizit die Default-Konstruktoren in ihren Klassen.

2. Umsetzen einer geeigneten Datenstruktur und weitere Konstruktoren

a) Sorgen Sie dafuer, dass `Motorrad, Bus, LKW und Auto` von `Fahrzeug` alle Methoden und Attribute erben. Erweitern Sie Ihre Klassen mit dem/den entsprechenden Schluesselwort, bzw. Schluesselwoertern.

b) Statten Sie die Klassen mit weiteren geeigneten Konstruktoren aus.Sorgen Sie dafuer, dass bei Benutzung der neuen Konstruktoren **alle** Variablen gefuellt werden, auch die der Superklasse.

3. Implementieren einer `Main-Methode` und in der Klasse `FahrzeugApp`

a) Erstellen Sie eine Klasse `FahrzeugApp` und statten Sie diese waehrend der Erzeugung oder manuel mit einer `Main-Methode` aus.

b) Erzeugen Sie in der `Main-Methode`

folgende Motoren:

- Variablenname „k4m“ und
  - Hubraum: 1598
  - Leistung: 110
  - Kraftstoff: Benzin
  - Nockenwellensteuerung: Zahnriemen
- Variablenname „om601“ und
  - Hubraum: 1997
  - Leistung: 72
  - Kraftstoff: Benzin
  - Nockenwellensteuerung: Steuerkette
- Variablenname „om542“ und
  - Hubraum: 14997
  - Leistung: 653
  - Kraftstoff: Diesel
  - Nockenwellensteuerung: Steuerkette
- Variablenname „jx21“ und
  - Hubraum: 998
  - Leistung: 175
  - Kraftstoff: Benzin
  - Nockenwellensteuerung: Stirnrad

folgende Fahrzeuge und passen Sie falls noetig das Format der Eingabewerte an:

- Auto mit Variablenname „renault“ und
  - Karosserieform: Kombi
  - Motor: k4m
  - Hoechstgeschwindigkeit: 190
  - Zulaessige Gesamtmasse in Tonnen: 1.590
  - Sitzplaetze: 5

- Auto mit Variablenname „mercedes“ und
  - Karosserieform: Limousine
  - Motor: om601
  - Hoechstgeschwindigkeit: 170
  - Zulaessige Gesamtmasse in Tonnen: 2.430
  - Sitzplaetze: 5

- LKW mit Variablenname „arctros“ und
  - Nutzlast: 20
  - Motor: om542
  - Hoechstgeschwindigkeit: 90
  - Zulaessige Gesamtmasse in Tonnen: 19.995
  - Sitzplaetze: 2

- Bus mit Variablenname „citaro“ und
  - Stehplaetze: 48
  - Motor: om542
  - Hoechstgeschwindigkeit: 105
  - Zulaessige Gesamtmasse in Tonnen: 12.590
  - Sitzplaetze: 50

- Motorrad mit Variablenname „ninja“ und
  - Kategorie: Supersport
  - Motor: jx21
  - Hoechstgeschwindigkeit: 390
  - Zulaessige Gesamtmasse in Tonnen: 310
  - Sitzplaetze: 2

c) Verpflichten Sie alle erbenden Klassen von `Fahrzeug` eine `toString-Methode` zu implementieren. Sorgen Sie dafuer, dass keine Objekte vom Typ Fahrzeug erstellt werden koennen. Kommentieren Sie notfalls nicht funktionsfaehigen Code wieder aus.

d) Zeigen Sie, dass im `arctros` und im `citaro` der gleiche Motor verbaut ist. Fragen Sie dazu den `arctros` und packen das in die Variable im `arctrosMotor`. Das selbe noch einmal mit dem `citaro` und packen das in die Variable `citaroMotor`. Speichern Sie das boolsche Ergebnis in die Variable `gleicherMotor`.

Kommentieren Sie desweiteren im Quellcode, ob sie die Objekte vergleichen oder den tatsaechlichen Inhalt der Variablen. Verdeutlichen Sie kurz den unterschied zwischen `==` und .equals.

e) Wegen dem Rasseln der Steuerkette erhaelt der `citaro` einen neuen Motor. Erstellen Sie dazu in der `Main-Methode` eine neue Variable mit dem Namen `om543`. Es aendert sich nur die Nockenwellensteuerung hin zu einem „Zahnriemen“.

4. Exceptions

Erstellen Sie eine Methode `pruefePassstrassentauglichkeit()` mit dem Return-Type `boolean`. Speichern Sie das Ergebnis der Abfragen in jeweile eine eigene Variable. Fuer jedes Fahrzeug, bei dem die zulaessige Gesamtmasse ueber 3,5 Tonnen liegt, soll eine `FahrzeugZuSchwer`**`Exception`** mit der Meldung „Fahrzeug fuer Passstrassen ungeeignet“ geworfen werden. Fangen Sie etwagige Exceptions in der `Main-Methode` ab.

5. Schnittstelle `Nutzfahrzeug`

Erstellen Sie ein Interface `Nutzfahrzeug` und statten Sie dieses mit einer Methode `getReturnOfInvestment()` aus, die einen `double` zurueckgibt. Sorgen Sie dafuer, dass LKW und Bus dieses Interface implementieren aber programmieren Sie die Methode falls möglich nur 1x aus. Geben Sie einen beliebigen `double`-Wert zurück.

6. Erstellen eines Fuhrparks mit Collections

a) Erstellen Sie eine Klasse `Fuhrpark`, die alle Objekte in einer Variable `objects` speichert. Nutzen Sie eine beliebige Collection.

b) erstellen Sie eine Variable `fahrzeuge` die nur Objekte vom typ `fahrzeuge` zulaesst.

c) implementieren Sie die Getter- und Settermethoden und ergaenzen Sie einen Konstruktor.

d) Erzeugen Sie in der `Main-Methode` der Klasse `FahrzeugApp` ein Object vom Typ `Fuhrpark`. Fuegen Sie alle bisher erstellten Objekte in `objects` ein und wiederholen Sie den Vorgang, indem Sie alle Fahrzeuge in die Variable `fahrzeuge` hinzufuegen.

e) Nutzen Sie eine `foreach-Schleife, Enumeration` oder einen `Interator` und geben Sie den kompletten Inhalt der Variable `fahrzeuge` auf die Konsole aus.

7. Comparable

a) Erstellen Sie in der `Main-Methode` der Klasse `FahrzeugApp` eine Variable `fuhrparkSortiertNachLeistung`. Fuegen Sie alle bisher erstellten Objekte aus `fuhrpark` in ein sortiertes Collection-Objekt `fuhrparkSortiertNachLeistung` ein (aehnlich wie Nr. 6 e, diesmal z.B. eine ArrayList). Sorgen Sie wieder dafuer, dass nur Fahrzeuge hinzugefuegt werden koennen. Programmieren Sie diese Methode nur 1x aus.

b) Sorgen Sie dafuer, dass **alle** Fahrzeuge `Comparable` sind. Sortieren Sie die Fahrzeuge nach Leistung.

Hochschule Ludwigshafen am Rhein

Programmieren II - Klausur SS2016

Bitte arbeiten Sie elektronisch Lösungen zu den folgenden Aufgaben auf dem zugewiesenen PC im Prüfungsraum aus. Erstellen Sie für Ihren Workspace einen Ordner „workspace“ auf dem Netzlaufwerk „P:/“.

Packen Sie Ihre Lösungen am Ende der Klausur in eine ZIP-Datei und nennen Sie diese Sxxxxxx-Klausur.zip (xxxxxx = Matr.Nr.). Speichern Sie die ZIP-Datei auf dem Laufwerk „P:/“, lassen Sie Eclipse geöffnet und den PC auch nach Ende der Klausur angeschaltet.

Achten Sie auf gut formatierten Code.

Aufgaben 1 -3 sind zu Anfang anzufertigen. Unter Umständen bietet Ihnen Eclipse an, diese in einem Rutsch zu lösen. Die anderen Aufgaben bauen darauf auf.

1 Erstellen der benötigten Klassen (6 Punkte)

a) Entwickeln Sie das Projekt `Raumfahrt` im Package `de.hs_lu.ss2016.raumfahrt`. Setzen Sie mit möglichst wenig Aufwand folgende Klassen um und achten Sie Augenmerk auf die Sichtbarkeit der Attribute:

`Raumfahrtausstattung`	
bezeichnung	Attribut vom Typ `String`
ablaufjahr	Attribut vom Typ `int` oder `Integer?`

(Achten Sie darauf, dass die „`ablaufjahr`“ im Fall, in dem sie nicht gesetzt wurde „`null`“ zurückgibt.)

`DockingPortSystem`	
entwicklungsjahr	Attribut vom Typ `int`

`Traegersystem`	
dockingPortSystem	Attribut vom Typ `DockingPortSystem`

`Raumstation`	
dockingPortSystem	Attribut vom Typ `DockingPortSystem`

`Rakete`	
kraftstoff	Attribut vom Typ `String`

b) Implementieren Sie explizit die parameterlosen Default-Konstruktoren in ihren Klassen.
c) Statten Sie die Klassen mit geeigneten Getter- und Settermethoden aus.

2 Umsetzen einer geeigneten Datenstruktur und weitere Konstruktoren (4 Punkte)

a) Sorgen Sie dafür, dass

- `DockingPortSystem`, `Traegersystem`, `Raumstation` und `Rakete` von `Raumfahrtausstattung`

alle Methoden und Attribute erben. Erweitern Sie Ihre Klassen mit dem/den entsprechenden Schlüsselwort, bzw. Schlüsselwörtern.

b) Statten Sie die Klassen mit weiteren geeigneten Konstruktoren aus. Sorgen Sie dafür, dass bei Benutzung der neuen Konstruktoren **alle** Variablen gefüllt werden, auch die der Superklasse.

3 Implementieren Sie eine `main-Methode` in der Klasse `RaumfahrtApp` (11 Punkte)

a) Erstellen Sie eine Klasse `RaumfahrtApp` und statten Sie mit einer `main-Methode` aus.

b) Erzeugen Sie in der `main-Methode` folgende Objekte:

- Ein `DockingPortSystem` mit Variablenname „`amerikanischesDockingPort`“ und
  - Bezeichnung: „`NASA-System`“
  - Ablaufjahr: „`2030`“
  - Ablaufjahr: „`1968`“
- Ein `DockingPortSystem` mit Variablenname „`russischesDockingPort`“ und
  - Bezeichnung: „`ROSKOSMOS-System`“
  - Ablaufjahr: „`1989`“
  - Ablaufjahr: „`1985`“
- Ein `Traegersystem` mit Variablenname „`spaceShuttle`“ und
  - Bezeichnung: „`Space Shuttle`“
  - DockingPortSystem: „`amerikanischesDockingPort`“
  - Ablaufjahr: „`2010`“
- Eine `Raumstation` mit Variablenname „`iSS`“ und
  - Bezeichnung: „`Internationale Raumstation`“
  - DockingPortSystem: „`amerikanischesDockingPort`“
  - Ablaufjahr: „`2030`“
- Ein `Traegersystem` mit Variablenname „`buran`“ und
  - Bezeichnung: „`Buran`“
  - DockingPortSystem: „`russischesDockingPort`“
  - Ablaufjahr: „`2004`“
- Eine `Raumstation` mit Variablenname „`mIR`“ und
  - Bezeichnung: „`MIR`“
  - DockingPortSystem: „`russischesDockingPort`“
  - Ablaufjahr: „`2001`“
- Eine `Rakete` mit Variablenname „`apollo`“ und
  - Bezeichnung: „`Apollo`“

- Kraftstoff: „fest“
- Ablaufjahr: „2030“

c) Aufgrund der Gefahren von potentiellen Sauerstoffeinschlüssen in festem Kraftstoff wird die Rakete auf „flüssig“ umgerüstet. Ändern Sie das nachträglich und setzen Sie die Variable mit der entsprechenden Methode.

d) Sorgen Sie dafür, dass keine Objekte vom Typ `Raumfahrtausstattung` erstellt werden können, da es sich bei `Raumfahrtausstattung` um einen Oberbegriff handelt. Kommentieren Sie notfalls nicht funktionsfähigen Code wieder aus!

e) Verpflichten Sie alle erbenden Klassen von `Raumfahrtausstattung` eine `toString-Methode` zu implementieren. Beheben Sie die Ursache der Complilerwarnungen. **Achten Sie dabei auf eine gut formatierte Ausgabe und nutzen sie in einer Ausgabe mindestens 2 besondere Maskierungen** für Ausgaben von Sonderzeichen. Es sollen alle Variablen des Objekts ausgegeben werden!

f) Erstellen Sie in der Klasse `RaumfahrtApp` eine Methode `isTheSame (Object object1, Object object2)`, welche einen `boolean` zurückgibt und geben Sie auf der Konsole (bitte den Aufruf an geeigneter Stelle OBJEKTORIENTIERT implementieren!!) aus, dass es sich bei den `DockingPortSystemen` von „`spaceShuttle`“ und der „`buran`“ um unterschiedliche Systeme handelt.

g) Erklären Sie kurz in einem Kommentar im Quellcode zur Methode der vorherigen Aufgabe `isTheSame` welche Möglichkeiten es zum Objektvergleich gibt und welche man in welchen Fällen anwendet, damit diese über alle JVMs hinweg das voraussagbare bzw. beeinflussbare Ergebnisse liefern.

4 Exceptions (11 Punkte)

a) Programmieren Sie eine Exception mit Name `ZuAltFuerErneutenEinsatzException`, welche im aufgerufenen Fall die Meldung „`Gerät zu alt für erneute Nutzung!`“ ausgibt.

Tipp: Zusätzliche Variablen und Methoden werden nicht zwingend benötigt. Warnungen bzgl. Serial Version ID können ignoriert werden!

b) Erstellen Sie in der Klasse `Raumfahrtausstattung` eine Methode `pruefeVerwendbarkeit()`. Prüfen Sie in der Methode, ob das Gerät noch verwendet werden darf, sprich, das `Ablaufjahr` nicht überschritten ist. Geben Sie im positiven Falle, dass das `Ablaufjahr` > 2016 ist einen String mit der `Bezeichnung` und “ - `OK`“ zurück. Andernfalls ***werfen*** Sie eine `ZuAltFuerErneutenEinsatzException`.

c) Implementieren Sie den Aufruf der `main` und prüfen Sie, ob „`buran`“ noch verwendet werden darf. Behandeln Sie die mögliche Exception dort und geben Sie im Fehlerfall nur die Message in roter Farbe auf die Konsole aus. Geben Sie das Ergebnis im positiven Fall ebenfalls auf die Konsole aus.

5 Schnittstelle `HasDockingPortSystem` (10 Punkte)

a) Erstellen Sie ein Interface `HasDockingPortSystem`. Sorgen Sie dafür, dass dieses Interface von `TraegerSystem` und `Raumstation` implementiert wird.

b) Verpflichten Sie alle Klassen, welche dieses Interfache implementieren, mit den `Gettern & Settern` für das `DockingPortSystem` ausgestattet zu sein.

c) Erstellen Sie in der `RaumfahrtApp` einen Kommentar und **begründen** Sie, wo ein Interface `NochFuerRaumfahrtGeeignet` mit der Methode `pruefeVerwendbarkeit()` sinnvollerweise implementiert werden sollte.

6 Erstellen eines Sortiments mit Collections (18 Punkte)

a) Erstellen Sie geeignete Klasse `Inventar`, welche das Interface `Collection` erfüllt.

b) Erstellen Sie eine Klasse `Raumfahrttechnologie`, die

- eine Variable „`traegerSysteme`“ vom Typ `Inventar` enthält und schränken Sie diese auf `TraegerSysteme` ein
- und eine Variable „`raumstationen`“ vom Typ `Inventar` enthält und schränken Sie diese auf `Raumstationen` ein

c) Implementieren Sie falls sinnvoll die Getter & Setter und sorgen Sie dafür, dass die Klassen `Inventar` und `Raumfahrttechnologie` einen passenden Konstruktor haben.

d) Sorgen Sie dafür, dass jedes Objekt nur einmal im Inventar zu finden ist. Schaffen Sie dazu eine programmatische Lösung.

e) Erstellen Sie in Ihrer `RaumfahrtApp` eine Variable „`nasaBestand`“ vom Typ „`Raumfahrttechnologie`“. Fügen Sie in

- „`traegerSysteme`“ die Variablen “`spaceShuttle`“
- „`raumstationen`“ die Variable „`iSS`“

ein und beweisen Sie, dass Sie in „`traegerSysteme`“ nicht die „`amerikanischesDockingPort`“ einfügen können.

f) Erstellen Sie sich eine neue Variable vom Typ `Inventar` und Namen `gesamteRaumfahrtausstattung`. Fügen Sie alle bisher erstellten Objekte in die Collection ein und nutzen Sie eine `foreach-Scheife` oder einen `Iterator` um ***nur das Entwicklungsjahr*** von den `DockingPortSystem`en auf die Konsole auszugeben.

7 Zusatzaufgabe `Comparable` (6 Zusatzpunkte)

a) Sorgen Sie dafür, dass Klassen, welche **Raumfahrtausstattung** sind, das Interface `Comparable` implementieren.

b) Erstellen Sie in der main eine ArrayList mit Namen `raumfahrtausstattung` und fügen Sie alle Objekte, welche **Raumfahrtausstattung** sind hinzu. Geben Sie diese auf die Konsole aus. Beheben Sie die Compilerwarnungen und programmieren Sie die Methode fachgerecht aus.

c) Sortieren Sie alle Objekte innerhalb von `raumfahrtausstattung` und geben Sie die ArrayList erneut auf die Konsole aus.

Viel Erfolg!