Prof. Dr. Stefan Göller
Florian Bruse
Dr. Norbert Hundeshagen

Einführung in die Informatik

WS 2018/2019

Übungsblatt 10
17.1.2019-24.1.2019

Abgabe: Bis zum 24.1. 18:00 Uhr über moodle. Reichen Sie pro Aufgabe, die Sie bearbeitet haben, die vorgegebene Rumpfdatei ein. Verändern Sie diese Datei nicht weiter als angegeben. Andere Dateien oder unzulässig veränderte Rumpfdateien werden im Zweifelsfall nicht korrigiert.

Aufgabe 1 (Objektorientierte Modellierung) (7*5=35 Punkte):

Wir befinden uns im Jahre 50 vor Christus. Ganz Gallien ist von den Römern besetzt. Das Zusammenleben von Galliern und Römern läuft sehr harmonisch. Beide Völker messen sich regelmäßig im sportlichen Wettkampf.

Weibliche Gallier haben Namen, die auf "ine" enden, männliche Gallier haben Namen, die auf "ix" enden. Gallier leben in Dörfern. Ein Dorf hat zwei herausragende Positionen: Den Druiden und den Barden, die beide männlich oder weiblich sein können. Ämterhäufung ist zulässig.

Weibliche Römer haben Namen, die auf "a" enden, männliche Römer haben Namen, die auf "us" enden.[1] Römer haben *einen* Imperator beliebigen Geschlechts und arbeiten in Legionen, welche aus einer Reihe von Soldaten besteht, die immer männlich sind. Jede Legion wird von einem Zenturio angeführt, der nicht männlich sein muss und selbst nicht Soldat dieser Legion ist. Gelegentlich sendet der Imperator eine Legion gegen ein gallisches Dorf aus. Es kommt dann zum oben erwähnten sportlichen Wettkampf, den die Gallier gewinnen, weil sie einen Zaubertrank haben. Der Druide nimmt aus Altersgründen nicht

[1] Der Name Caesars ist Gaius Julius.

am Wettkampf teil. Anschließend gibt es auf Seiten der Gallier ein Bankett, in dem jeder Bewohner ein Wildschwein isst. Davon ausgenommen ist der Barde, der sicherheitshalber an einen Baum gebunden wird.

Setzen Sie nun die obigen Sachverhalte mittels objektorientierter Programmierung um. In der Rumpfdatei zu dieser Aufgabe finden Sie eine Definition der Klasse `Mensch`, welche die Attribute `name` (eine Zeichenkette) und `weiblich` (ein `bool`) hat. Beachten sie, dass die Klasse keine Setter-Methode für das Attribut `weiblich` hat, weil das inhaltlich nicht sinnvoll ist. Lösen Sie die Aufgabe vollständig innerhalb dieser Rumpfdatei. Ihr Code für Aufgabenteil g) dient als Ergebniskontrolle und soll, genauso wie der vorgegebene Testcode, fehlerfrei (insbesondere ohne Exceptions und Fehlermeldungen) durchlaufen.

a) Erstellen sie zunächst eine Klasse `Gallier`, welche von `Mensch` erben soll. Instanzen der Klasse Gallier haben ein zusätzliches Attribut, welches angibt, wie viele Wildschweine der betreffende Gallier gegessen hat. Die Methode `get_wildschweine(self)` soll den Wert dieses Attributs zurückgeben, mit der Methode `iss_wildschwein(self)` soll der betreffende Gallier ein Wildschwein essen.

b) Erstellen Sie eine Klasse `Roemer`, welche auch von `Mensch` erbt. Instanzen dieser Klasse haben ein zusätzliches Attribut, welches speichert, wie oft der betreffende Römer einen Wettkampf verloren hat. Mit diesem Attribut soll mittels der beiden Methoden `verliere(self)` und `wie_oft_verloren(self)` interagiert werden. Zusätzlich hat die Klasse `Roemer` die Klassenvariable `imperator` vom Typ `Roemer`, welche speichert, wer Imperator ist. Mit der Methode `werde_imperator(self)` soll ein `Roemer` zum Imperator werden können. Stellen Sie sicher, dass es immer einen Imperator gibt, sobald mindestens ein `Roemer` existiert.

c) Setzen Sie für die Klassen `Gallier` und `Roemer` die Namenskonventionen der beiden Völker durch, indem Sie in den Initialisierungsfunktionen und Getter-Methoden, wenn nötig, die passenden Namensendungen an einen gewünschten Namen anhängen. Soll beispielsweise ein männlicher `Gallier` mit dem Namen `"Un"` erstellt werden, verändern Sie den Namen zu `"Unix"`. Soll der Name einer `Roemer`in nach `"Lil"` geändert werden, heisst sie danach `"Lila"`. Passende Namen werden nicht verändert: eine `Gallier`in, der `"Hermine"` heissen soll kann auch so heißen.

d) Erstellen Sie eine Klasse `Dorf`, welche Attribute `bewohner` (eine Menge (`Set`) von `Galliern`) sowie `druide` und `barde` (jeweils `Gallier`) haben, und auch in dieser Reihenfolge an die Initialisierungsfunktion übergeben werden. Erstellen Sie für die Attribute `druide` und `barde` Getter- und Setter-Methoden, und beachten Sie, dass Druide und Barde auch Bewohner sind. Erstellen Sie für das Attribut `bewohner` eine Getter-Methode.

e) Erstellen Sie weiterhin eine Klasse `Legion`, welche Attribute `soldaten` (eine Menge (`Set` von `Roemern`) und `zenturio` (ein `Roemer`) hat, die in dieser Reihenfolge an

die Initialisierungsfunktion übergeben werden. Das Attribut `zenturio` soll normale Getter- und Setter-Methoden bekommen, das Attribut `soldaten` soll mittels der Methoden `rekrutiere(self, ein_roemer)` bzw. `pensioniere(self, ein_roemer)` manipuliert werden. Das zweite Argument ist dabei jeweils vom Typ `Roemer`. Beachten Sie die Regeln dafür, wer Soldat sein kann–wer nicht Soldat sein kann, wird nicht rekrutiert.

f) Schreiben Sie schließlich eine Funktion `wettkampf(ein_dorf, eine_legion)`, welche einen sportlichen Wettkampf zwischen ihren beiden Argumenten simuliert und zeilenweise (In der Form `"Gallier XYine misst sich mit Römer/Zenturio ABCus"`) ausgibt, welcher Gallier sich mit welchen Römern misst. Die genaue Aufteilung der Römer auf die Gallier bleibt Ihnen überlassen, es soll aber, sofern genug Römer da sind, jeder geeignete Gallier mindestens einen Römer abbekommen. Beachten Sie, dass sich jeder Römer pro Wettkampf mit genau einem Gallier misst, dass aber der Zenturio auch teilnimmt. Vergessen Sie anschließend das Wildschweinbankett nicht.

g) Fügen Sie ihren Testcode an der gekennzeichneten Stelle in der Datei ein. Erstellen Sie die geforderten Instanzen in der angegebenen Reihenfolge.

- Erstellen Sie `Gallier`innen `Laureline`, `Canine` und `Apfelsine` in gleichnamigen Variablen, wobei Sie der Initialisierungsfunktion für `Laureline` nur die Zeichenkette `"Laurel"` übergeben.
- Erstellen Sie `Gallier` `Praefix`, `Infix` und `Postfix` in gleichnamigen Variablen, wobei Sie bei letzterem nur die Zeichenkette `"Postf"` übergeben.
- Erstellen Sie `Roemer`innen `Salta`, `Mendoza` und `Ushuaia` in gleichnamigen Variablen, wobei Sie der Initialisierungsfunktion für `Salta` nur die Zeichenkette `"Salt"` übergeben.
- Erstellen Sie `Roemer` `Primus`, `Secundus`, `Tertius`, `Quartus` und `Quintus` in gleichnamigen Variablen, wobei Sie bei letzterem nur die Zeichenkette `"Quint"` übergeben.
- Erstellen Sie ein `Dorf` `Oelixdorf` mit den Bewohnern `Laureline`, `Apfelsine` und `Praefix`. Der Druide ist `Laureline`, der Barde ist `Praefix`. Erstellen Sie ein Dorf `Bekdorf` mit den Bewohnern `Laureline`, `Postfix` und `Infix`. Der Druide ist `Infix`, der Barde ist `Canine`.
- Erstellen Sie Legion `Hispana` mit Zenturio `Salta` und Soldaten `Quintus, Quartus, Tertius, Mendoza`.

Möge Ihnen der Himmel nicht auf den Kopf fallen.

Aufgabe 2 (Spezielle Operatoren) (4+4+4+5+4+4=25 Punkte):

a) Definieren Sie eine Klasse `Intliste`, welche Listen modelliert, die nur aus Integern bestehen. Instanzen der Klasse `Intliste` soll ein Attribut `liste` haben, welches aus einer solchen Liste aus Integern besteht. Schreiben Sie die Getter- und Settermethoden für dieses Attribut und die Initialisierungsfunktion der Klasse so, dass Sie eine beliebige Liste entgegennehmen, und nur die Teilliste der darin enthaltenen Integer in das Attribut `liste` schreiben. Beispielsweise erzeugt `Intliste([1,11,"111"])` eine `Intliste` mit den Einträgen 1 und 11, aber ohne die Zeichenkette `"111"`.

b) Implementieren Sie für die Klasse `Intliste` die Funktion `__len__(self)` so, dass die Länge einer `Intliste` gerade die Länge der von Ihr gespeicherten Liste ist. Beispielsweise soll `len(Intliste([1,11]))` zu 2 auswerten.

c) Implementieren Sie für die Klasse `Intliste` die Funktion `__str__(self)` so, dass die Integer bei der Ausgabe durch Auffüllen mit führenden Nullen alle die gleiche Stelligkeit haben. Beispielsweise soll das Kommando `print(Intliste([1,11,"111"]))` die Ausgabe `[01,11]` erzeugen. Das Kommando `print(Intliste([15,7.2,332, "x"]))` soll die Ausgabe `[015,332]` erzeugen.

d) Implementieren Sie für die Klasse `Intliste` die Funktionen `__eq__(self, other)`, `__ne__(self, other)`, `__lt__(self, other)`, `__le__(self, other)`, `__gt__(self, other)` und `__ge__(self, other)` so, dass für zwei `Intlisten` mit Namen `links` und `rechts` gilt, dass

- `links` und `rechts` genau dann als gleich angesehen werden, wenn Sie die gleichen Einträge in der gleichen Reihenfolge in ihrer jeweiligen Liste speichern,
- `links != rechts` genau dann zu `True` auswertet, wenn die beiden Listen nicht die gleichen Einträge in der gleichen Reihenfolge enthalten,
- `links < rechts` genau dann zu `True` auswertet, wenn `links` lexikographisch[2] kleiner als `rechts` ist (Beispielsweise soll `Intliste([3,1]) < Intliste([3,2])` gelten, und auch `Intliste([[3,1]) < Intliste([3,1,4])`,aber nicht gelten soll `Intliste([3,1,4]) < Intliste([3,1])`),
- `links > rechts`, `links <= rechts` und `links >= rechts` so auswerten, wie man es von einer totalen Ordnung erwarten würde. Beispielsweise soll `Intliste(3,1) <= Intliste(3,1)` und `Intliste([3,1]) < Intliste([3,1,4])` gelten, aber nicht `Intliste([3,1]) > Intliste([3,1])`.

[2] Für zwei Listen `L` und `M` gilt, dass `L` lexikographisch kleiner ist als `M` falls `L[i] < M[i]` wobei i die erste Position ist, an der sich die beiden Listen unterscheiden, oder falls `L` kürzer ist als `M` und und `L[i] == M[i]` für alle i für welche `L` Einträge hat.

e) Implementieren Sie für die Klasse `Intliste` die Funktion `__add__(self, other)`. Dabei soll für zwei `Intlisten`, beispielsweise mit den Namen `links` und `rechts`, der Ausdruck `links + rechts` eine `Intliste` erzeugt, die erst alle Einträge der Liste von `links` und dann alle Einträge der Liste von `rechts` enthält. Beispielsweise soll `print(Intliste([3,1]) + Intliste([4,1,5])` die Ausgabe `"[3,1,4,1,5]"` erzeugen.

f) Implementieren Sie für die Klasse `Intliste` die Funktion `__sub__(self, other)`. Dabei soll für zwei `Intlisten`, beispielsweise mit den Namen `links` und `rechts`, der Ausdruck `links - rechts` eine `Intliste` erzeugen, die nur die Integer aus der Liste von `links` enthält, die nicht in der Liste von `rechts` vorkommen. Die Reihenfolge soll sich dabei nicht ändern. Beispielsweise soll `print(Intliste([3,1,4,1]) - Intliste([1]))` die Ausgabe `"[3,4]"` erzeugen.