Lehrstuhl für Angewandte Informatik I
Prof. Dr. Bernhard Westfechtel
Sandra Greiner

WS 2017/18
11. Januar 2018

Konzepte der Programmierung

Übung 11 – Vererbung und Schnittstellen

Abgabe: 25.01.2018

Aufgabe 11.1 (Vererbung)

Die folgende Aufgabe übt zentrale Konzepte der Vererbung und damit der objektorientierten Programmierung ein. Beachten Sie die Unterschiede zwischen abstrakten und „normalen“ Klassen.

Wir implementieren ein Grundgerüst für die Verwaltung von Sportmannschaften. Implementieren Sie dafür

(a) als Erstes eine **offene Datentypklasse** `Datum`, die sich aus Feldern für den Tag (Ganzzahl), den Monat und das Jahr zusammensetzt. Warum brauchen Sie in dieser Klasse keinen Konstruktor? Geben Sie die Antwort im Quelltext der Klasse.

(b) eine **abstrakte Klasse** `Sportler`. Eine Sportler besitzt die folgenden **für Subtypen sichtbare** individuelle Eigenschaften: einen Namen und ein Geburtsdatum. Diese Eigenschaften werden im **Konstruktor** gesetzt. Außerdem ist in der Klasse eine **abstrakte Objektmethode** `istTeammitglied(...)` definiert.

Redefinieren Sie die `toString()`-Methode (implizit geerbt von `java.lang.Object`). Sie soll den Namen und den Geburtstag ohne die Jahreszahl zurückgeben.

(c)
- eine konkrete Klasse `FussballSpieler`, die von `Sportler` **erbt**. Sie speichert für jeden Spieler in einem **für Subtypen sichtbarem Objektfeld**, ob er sich gerade in einer Mannschaft befindet. Dieses Feld wird im Konstruktor mit `false` initialisiert und soll über eine setzte-Methode veränderbar und über die Methode `istTeammitglied(...)` abfragbar sein. Der Name und das Geburtsdatum werden durch Weitergabe geeigneter Parameterwerte an den **super-Konstruktor** gesetzt (Gebrauch von *Wiederverwendung*).

  Außerdem gibt es eine Methode `istTorwart(...)`. Ein Spieler ist zunächst kein Torwart.

- sowie eine Klasse `Torwart` soll sich wie `FussballSpieler` verhalten. Folgende Unterschiede gibt es:
  - Die `toString(...)`-Methode soll mit der Zeichenkette „`(Torwart)`“ beginnen. **Redefinieren** Sie die Methode und **delegieren** Sie dabei zur `toString(...)`-Methode der Oberklasse. Vergessen Sie außerdem nicht, den **Konstruktor** explizit zu erben.
  - Die Methode `istTorwart(...)` gibt `true` zurück.

(d) ein `FussballTeam`, das sich aus einem Torwart und 10 Feldspielern zusammensetzt. Speichern Sie die Feldspieler in einem privaten Objektfeld, für das Sie ausreichend Speicher allokieren.

Torwart und Teamname werden ebenfalls jeweils in einem **privaten** Objektfeld gespeichert und beide im Konstruktor gesetzt. Ändern Sie darin auch den Status der Teammitgliedschaft des Torwarts.

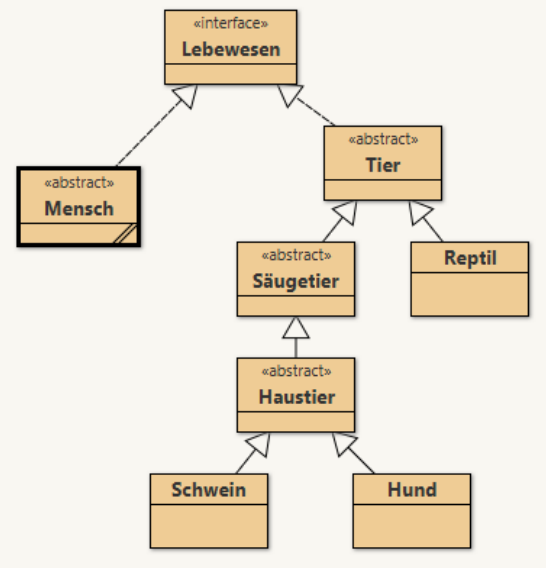
Der Zugriff auf das Spieler-Array soll ausschließlich durch folgende Methoden erfolgen:

- `gibSpieler(...)`: Soll eine **flache Kopie**[1] des Spieler-Arrays zurückgeben.
- `registriereSpieler(...)`: Ihr wird ein neuer Spieler übergeben. Sie soll zunächst prüfen, ob der übergebene Spieler ein Torwart ist. Nur wenn dies nicht der Fall ist und das Spieler-Array zusätzlich noch nicht voll ist, soll der übergebene Spieler an der nächstmöglichen Position im Spieler-Array des Teams abgelegt werden. Ist dies der Fall wird auch von diesem Spieler der Mitgliedsstatus geändert.

Implementieren Sie außerdem wieder die `toString(...)`-Methode, sodass Teamname, Torwart und alle Mitglieder zeilenweise zurückgegeben werden.

(1+2+4+3 = 10 Punkte)

Aufgabe 11.2 (Vererbungshierarchien und dynamische Typbestimmung)
Bearbeiten Sie diese Aufgabe handschriftlich.
Gegeben sei die folgende Vererbungshierarchie (grafisch in BlueJ dargestellt):



Sowie folgendes Java-Quelltextfragment:

```
1     Lebewesen l = new Hund();
2     Tier t = new Reptil();
3     Haustier ht = l;
4     Säugetier s1 = new Schwein();
5     Hund h = new Hund();
6     Schwein s2 = new Schwein();
7     s1 = ht;
8     ht = h;
```

[1] *Flache Kopie* bedeutet, dass die Inhalte des ursprünglichen Arrays nicht neu erzeugt, sondern lediglich referenziert werden.

(a) Geben Sie einen Java-Quelltext an, der die Vererbungshierarchie erzeugt.

(b) Schlangen sind Reptilien, die mitunter als Haustiere gehalten werden. Warum ist es nicht möglich, diesen Sachverhalt explizit in der Vererbungshierarchie darzustellen? Wie könnte man dieses Problem lösen?

(c) In welcher Zeile des gegebenen Quelltextfragments werden die Regeln der **Substituierbarkeit** verletzt? Wie könnte man das Problem lösen, ohne existierenden Quellcode zu streichen?

(d) Geben Sie den **statischen** sowie den **dynamischen** Typ der folgenden Variablen **nach** den folgenden Quelltextzeilen an: `s1` (Z. 4 und nach Z.7), `t` (Z. 2), `ht` (Z. 3, Z. 7 und nach Z. 8).

Erklären Sie anschließend mit eigenen Worten den Unterschied zwischen **statischem** und **dynamischem** Typ.

(**3+1+1+3 = 8 Punkte**)

Aufgabe 11.3
Geben Sie für folgende Konzepte ein eigenes Beispiel und anhand dessen eine kurze Erläuterung des Konzepts mit eigenen Worten an.
Bearbeiten Sie die Aufgabe *handschriftlich*.

- Unterschied zwischen **statischem** und **dynamischem** Binden
- Deklaration eines Feldes oder einer Methode als `final`

(**3 Punkte**)

Aufgabe 11.4 (Doppelt-verkette Listen)
Auf dem letzten Übungsblatt sollten Sie eine doppelt-verkette Liste für Plätzchendosen (`Dosenliste`) schreiben.

Wir wollen jetzt auch Songs in doppelt-verketteten Listen speichern können. Die Klasse Song sieht wie folgt aus:

```
1 public class Song {
2     private String titel;
3     private String interpret;
4 }
```

(a) Erweitern Sie zunächst die Klasse Song um einen geeigneten Konstruktor, der die Felder mit Werten belegt. Redefinieren Sie die `toString(...)` Methode, sodass Sie die Daten des Songs formatiert zurückgibt.

(b) Implementieren Sie nun mithilfe der Lösung des letzten Blattes (`Dosenliste`) eine `Playlist` die aus `Element`en besteht, die Songs referenzieren. In der Playlist soll eine Methode `play(...)` hinzugefügt werden, die alle Songs, die in der Liste gespeichert sind, unter Nutzung von `toString(...)` der Reihe nach auf der Konsole ausgibt. Ihr wird als Parameter ein Wert übergeben, der angibt, ob die Lieder in einer Endlosschleife gespielt werden sollen. Ist dies der Fall, wird nach dem letzten Listeneintrag von vorne begonnen. Überlegen Sie sich für diesen Fall ein geeignetes Abbruchkriterium.

(**3 Punkte**)

Aufgabe 11.5 (Interfaces)
In dieser Aufgabe sollen Sie mit Hilfe eines **Interfaces** eine Schnittstelle für die Datenstruktur **Schlange** (vgl. Kapitel Datenstrukturen, Folien 13 bis 23) mit Eintragstyp `Double` definieren, die durch zwei unterschiedliche Realisierungen – als Array und als einfach-verkettete Liste – implementiert wird. Man könnte damit zum Beispiel Noteneinträge von Studenten verwalten.

(a) Erstellen Sie ein neues **Interface** `Schlange`, das folgende **Objektmethoden** definiert:

- `reiheEin(...)`: Reiht einen übergebenen `Double` am Ende der Schlange ein und gibt im Erfolgsfall `true` zurück.
- `entferne(...)`: Falls die Schlange nicht leer ist, wird der Double am Anfang der Schlange entfernt und zurückgegeben, ansonsten wird `null` zurückgegeben.

(b) Schreiben Sie eine Klasse `ListeSchlange`, die das Interface `Schlange` mit Hilfe einer **einfach-verketteten Liste** implementiert. Beachten Sie, dass die Doubles nicht direkt in der Liste gespeichert werden sollen.

(c) Schreiben Sie eine Klasse `ArraySchlangeZyklisch`, die das Interface `Schlange` mit Hilfe eines **zyklischen Arrays** implementiert. Im Konstruktor soll die maximale Länge der Schlange angegeben werden.

(d) Nun soll ein weiteres Interface `Laengenbeschraenkung` eingeführt werden, das eine **Objektmethode** `gibMaximaleLaenge(...)` definiert, die die maximale Länge der Datenstruktur zurückgibt.

(e) Ein drittes Interface `SchlangeMitLaengenbeschraenkung` soll alle Eigenschaften der beiden Interfaces `Schlange` und `Laengenbeschraenkung` zusammenfassen.

(f) Kopieren Sie die Klasse `ArraySchlangeZyklisch` aus Aufgabe c in eine neue Klasse `ArraySchlange`. Ändern Sie darin die Implementierung aus c so ab, dass `ArraySchlange` das Interface `SchlangeMitLaengenBeschraenkung` implementiert. In der `reiheEin(...)`-Methode soll zusätzlich geprüft werden, ob durch das Einfügen die Längenbeschränkung verletzt wird. In diesem Fall soll der Einfügevorgang abgebrochen und `false` zurückgegeben werden.

(**2+4+4+1+1+1=13 Punkte**)

Aufgabe 11.6 (Bonus:Java-Klassen zur Ein- und Ausgabe)
Systematisieren Sie die folgenden Java-Klassen aus dem Paket *java.io*:

Writer, Reader, OutputStream, InputStream, OutputStreamWriter, InputStreamReader, FileOutputStream, FileInputStream, FileWriter, FileReader, BufferedWriter und BufferedReader

D.h. stellen Sie dar, welche Klassen von einander abgeleitet sind und geben Sie zu jeder Klasse ein kurze Beschreibung an. Verwenden Sie evtl. die Java-API-Dokumentation dazu.

(2 **Punkte**)