Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Peer Kröger
Janina Bleicher, Florian Richter

München, 12.01.2017

Einführung in die Programmierung

WS 2016/17

Übungsblatt 11: Exceptions, Interfaces

Besprechung: 23./25./27.01.2017

Ende der Abgabefrist: Freitag, 20.01.2017 14:00 Uhr.

Geben Sie Ihre Lösung in **Zweierteams** ab.
Der Javacode muss kompilieren. Löschen Sie jegliche packages aus Ihrer Abgabe heraus, sofern nicht anders verlangt.
Außerdem soll jeglicher Code kommentiert sein, sodass ersichtlich ist, was die Methoden machen.

Aufgabe 11-1 *Überschreiben, Überladen, Verdecken* **2+4+3 Punkte**

Betrachten Sie die folgenden Klassen:

```
1  public class A {
2    public int i = 11;
3    public A(int i) {                    // Signatur: A(I)
4      super();
5      this.i = i;
6    }
7
8    public A(float f) {                  // Signatur: A(F)
9      this((int)(f + 1));
10   }
11
12   public void f(int i, A a) {}         // Signatur: A.f(IA)
13   public void f(long lo, B b) {}       // Signatur: A.f(LB)
14   public void f(long lo, A a) {}       // Signatur: A.f(LA)
15 }
```

```
1  public class B extends A {
2    public float f = 31;
3    public B(double d) {                 // Signatur: B(D)
4      this((float)(d - 1));
5    }
6    public B (float f) {                 // Signatur: B(F)
7      super(f);
8      this.f = f;
9    }
10   public void f(int i, A a) {}         // Signatur: B.f(IA)
11   public void f(int i, B b) {}         // Signatur: B.f(IB)
12   public void f(long lo, A a) {}       // Signatur: B.f(LA)
13 }
```

```
1  public class C {
2    public static void main (String [] args) {
3      A a1 = new A(31);                                    // (K1)
4      System.out.println("A.i: " + a1.i);
5      A a2 = new A(31.9999f);                              // (K2)
6      System.out.println("A.i: " + a2.i);
7      A ab = new B(3.5);                                   // (K3)
8      System.out.println("A.i: " + ((A) ab).i);
9      System.out.println("B.i: " + ((B) ab).i);
10     B b = new B(8);                                      // (K4)
11     System.out.println("A.i: " + ((A) b).i);
12     System.out.println("B.i: " + ((B) b).i);
13
14     int i = 1;
15     long lo = 2;
16     a1.f(i, ab);                    // (M1)
17     a1.f(lo, a1);                   // (M2)
18     a1.f(lo,b);                     // (M3)
19
20     b.f(i,b);                       // (M4)
21     b.f(i, ab);                     // (M5)
22     b.f(lo,ab);                     // (M6)
23     b.f(lo, b);                     // (M7)
24
25     ab.f(i, a1);                    // (M8)
26     ab.f(lo, ab);                   // (M9)
27     ab.f(lo,b);                     // (M10)
28   }
29 }
```

Geben Sie an, welche Konstruktoren- und Methodenaufrufe stattfinden. Benutzen Sie nach Möglichkeit keinen Computer, sondern nutzen Sie Ihr Wissen zur Objektorientierung aus der Vorlesung. Verwenden Sie die als Kommentare gegebenen Funktionssignaturen. Begründen Sie alle Antworten kurz.

(a) Beschreiben Sie in maximal 3 Sätzen den Unterschied zwischen Überschreiben, Überladen und Verdecken.

(b) Geben Sie an, welche Konstruktoren in welcher Reihenfolge an den mit (K1)-(K4) bezeichneten Stellen in Klasse `C` benutzt werden. Achtung: Wie Sie wissen, hat jede Klasse außer `Object` eine Oberklasse. Geben Sie außerdem an, welche Attribute mit welchen Werten belegt werden und welche Werte durch die `plintln`-Anweisungen ausgegeben werden.

(c) Geben Sie für die mit (M1)-(M9) bezeichneten Stellen an, welche Funktionsvariante verwendet wird. Nutzen Sie dafür die vorgegebenen Signaturen.

Aufgabe 11-2 *Quartett* **5 Punkte**

In dieser Aufgabe modellieren Sie die Erstellung eines Quartetts. Ihnen steht bereits eine main-Methode zur Verfügung. Wie Sie sicherlich wissen, besteht ein Quartett aus maximal 4 Musikern. Implementieren Sie die Klasse `Musiker`, die maximal 4 Musiker unterstützt. Nutzen Sie dazu die Möglichkeiten der Datenkapselung. Wichtig ist hierbei, dass keine Musikerobjekte über die ersten 4 hinaus erstellt werden dürfen. Das heißt, dass auch keine andere Klasse den Konstruktur aufrufen können darf. Wenn das Maximum an Musikern erreicht ist, soll eine Exception geworfen werden, die eine entsprechende Meldung ausgibt.

Aufgabe 11-3 *Dungeons, Beasts and Swords (DBS)* **2+3+2+1+1+1+1 Punkte**

Modellieren Sie ein rudimentäres Spiel, in dem Menschen und Zwerge gemeinsam gegen Orks und Goblins kämpfen. Die Klasse `Kampf` soll dabei einen Kampf modellieren. Ein Kampf generiert ein Array aus n Einheiten, die dann gegeneinander ihren Kampf austragen. Eine `Einheit` ist eine abstrakte Klasse, die ein ganzzahliges Attribut `Lebenspunkte` enthält. Fällt dieser Wert im Laufe des Kampfes auf 0, so scheidet diese Einheit aus. Jeder beginnt mit 20 Lebenspunkten den Kampf. Eine Einheit kann ein `Mensch`, ein `Zwerg`, ein `Ork`, ein `Goblin` oder ein `Schaf` sein. Das Interface `Krieger` soll die Signatur der Methode `boolean kannAngreifen(Einheit ziel)` enthalten. Außer dem Schaf können alle anderen Einheiten attackieren, sind also Krieger. Um das Ziel einer Attacke zu bestimmen, soll das Array ab der Position des Angreifers durchlaufen werden. Nach dem letzten Element soll wieder an den Anfang gesprungen werden und das Array bis zum Angreifer weiter durchlaufen werden. Gültige Ziele für Menschen und Zwerge sind Orks und Goblins. Analog gilt, dass Orks und Goblins nur Menschen und Zwerge angreifen. Das erste gefundene gültige Ziel soll attackiert werden. Wurde eine Attacke ausgeführt, darf die nächste Einheit im Array attackieren. Zusätzlich greift jeder Schafe an, wenn sie im Weg stehen. Findet sich kein gültiges Ziel mehr, hat die attackierende Fraktion gewonnen. Dies sollte ausgegeben werden.
Bei einem Angriff sollen in der Regel 2 Lebenspunkte beim Ziel abgezogen werden. Es gibt allerdings 3 Eigenschaften, die die Attacken noch modifizieren: Menschen, Zwerge und Orks besitzen `SchwereRuestung`, die den Schaden auf sie selbst halbiert. Menschen, Zwerge und Goblins haben `Fernkampf`, die den Schaden um 2 Punkte erhöht. Außerdem haben Goblins `Gift` auf ihren Waffen, das ebenfalls nochmal den Schaden um 2 Punkte erhöht und nicht durch schwere Rüstung beeinträchtigt wird. Diese drei Eigenschaften sollen durch Markerinterfaces implementiert werden.

(a) Vervollständigen Sie in `Kampf.java` die Methode `findeNaechstesZiel(...)`. Diese soll prüfen, ob die Einheit an der angegebenen Position attackieren darf und dann das nächste gültige Ziel finden. Dies geschieht, in dem das Array vom Angreifer aus einmal durchlaufen wird. Wird ein passender Gegner gefunden, soll dieser zurückgegeben werden. Falls kein Ziel nach einer kompletten Iteration gefunden werden konnte, soll der Angreifer selbst als Rückgabewert verwendet werden. Falls der Angreifer nicht gültig ist, soll null zurückgegeben werden. In allen anderen Fällen liefert die Methode das Ziel des Angriffs.

(b) Implementieren Sie die abstrakte Klasse `Einheit`. Jede Einheit hat standardmäßig 20 Lebenspunkte. Fügen Sie eine Methode `kannAngreifen(Einheit ziel)` hinzu, die für Einheiten immer false zurückgibt. Außerdem soll eine Methode `werdeAngegriffen(int schaden)` die empfangenen Schadenspunkte verrechnen. `boolean lebtNoch()` soll widergeben, ob die Einheit noch über positiv viele Lebenspunkte verfügt. Schließlich wickelt die Methode `attackiere(Einheit ziel)` das eigentliche Angreifen ab.

(c) Implementieren Sie alle anderen oben genannten Einheiten-Unterklassen `Mensch`, `Zwerg`, `Ork`, `Goblin`, `Schaf`.

(d) Implementieren Sie das Kriegerinterface `Krieger` und stellen Sie sicher, dass es von den entsprechenden Klassen implementiert wird. Alle Krieger müssen wissen, wen sie angreifen dürfen. Dazu definieren Sie eine Interface-Methode `public boolean kannAngreifen(Einheit ziel)` im Kriegerinterface.

(e) Implementieren Sie Markerinterfaces `SchwereRuestung`, `Fernkampf`, `Gift`.

(f) Wenn ein Schaf angegriffen wird, soll eine SchafException geworfen werden und als entsprechenden Text ausgegeben werden. Implementieren Sie also eine entsprechende Exception-Klasse und überschreiben Sie die Methode `werdeAngegriffen(int schaden)` in der Schafklasse.

(g) Fügen Sie der Einheitenklasse ein Attribut Initiative hinzu. Dieses Attribut soll mit einem zufälligen Wert zwischen 0 und 100 im Konstruktor belegt werden. Sortieren Sie vor Kampfbeginn das Array nach den Initiativen, sodass die Einheit mit der höchsten Initiative beginnt. Implementieren Sie dazu das Interface `Comparable<Type>` in der Einheitenklasse und nutzen Sie `Arrays.sort()`.