technische universität
dortmund

Tutorial zu

Einführung in die Informatik für LogWIngs und WiMas

Wintersemester 2015/16

1 Zauberer und Zwerge, Aufgabenteil 1

Stellen Sie sich vor, Sie sollen eine Charakterverwaltung für das neue Onlinerollenspiel "Hallways and Hobbits" programmieren. Dazu benötigen Sie eine Klasse, die die Spielfiguren (Charaktere) modelliert. Schreiben Sie eine abstrakte Klasse *Charakter* mit den folgenden Eigenschaften:

1. Sie besitzt ein öffentliches Attribut *Name* vom Typ *String*, in dem der Name des Charakters gespeichert wird.
2. Sie besitzt ein nach außen geschütztes, aber vererbbares, ganzzahliges Attribut *Lebenspunkte*, in dem die Lebenspunkte des Charakters verwaltet werden.
3. Sie besitzt ein nach außen geschütztes, aber vererbbares, ganzzahliges Attribut *Erfahrungspunkte*. Bei der Erstellung eines neuen Charakters soll dieser noch keine Erfahrungspunkte besitzen. Initialisieren sie die Variable entsprechend.
4. Die Klasse besitzt eine öffentliche, nicht abstrakte Methode *erhalteErfahrung*, mit der die Erfahrungspunkte um einen übergebenen, ganzzahligen Wert erhöht werden können.
5. Die Stufe eines Charakters berechnet sihc mit der Formel: $Stufe = 1 + Erfahrungspunkte/100$. Fügen sie der Klasse *Charakter* eine öffentliche, nicht abstrakte Methode *gibStufe* hinzu, die die Stufe des Charakters berechnet und als ganzzahligen Wert zurückgibt.
6. Sinken die Lebenspunkte eines Charakters auf null oder weniger, so stirbt er. Dies sollen Sie mit Hilfe einer nicht abstrakten, öffentlichen Methode *verliereLebenspunkte* modellieren. Diese Methode erhält einen ganzzahligen Parameter, der von den Lebenspunkten des Charakters abgezogen wird. Sinken die Lebenspunkte dadurch auf null oder weniger, soll eine Nachricht auf dem Bildschirm im Format "Name ist gestorben" ausgegeben werden, wobei *Name* aus dem Attribut der Klasse Charakter stammt.

Zu Beginn der Bearbeitung einer Aufgabe sollte die Aufgabe immer vollständig gelesen werden. Die einzelnen Teilaufgaben sind normalerweise immer gleich aufgebaut. Anhand von bestimmten Schlüsselwörten kann entschieden werden, ob eine neue Klasse, eine Methode oder ein Attribut angelegt werden sollen. In diesem Fall soll eine "abstrakte Klasse *Charakter*" erstellt werden. Die Lösung dazu wäre also:

```
abstract class Charakter {

}
```

Diese Klasse soll das Grundgerüst eines Charakters modellieren. In einem Rollenspiel gibt es normalerweise verschiedene Arten von Charakteren, die sich gewisse Eigenschaften teilen. Die Erstellung einer abstrakten Klasse hat den Vorteil, dass diese Eigenschaften nicht für jede Art von Charakter neu erstellt werden müssen, sondern in einer Klasse vorhanden sind, von der später geerbt werden kann. Weiterhin ist es nicht möglich eine Instanz einer abstrakten Klasse zu erstellen. Dadurch wird verhindert, dass ein normaler Charakter ohne weitere Eigenschaften existiert.
Die Eigenschaften, die von alle verschiedenen Charakteren geteilt werden, werden in den weiteren Teilaufgaben modelliert.

In den Teilaufgaben 1-3 werden dem Klassengrundgerüst nun drei Attribute hinzugefügt. Ein Attribut wird folgendermaßen erstellt:

```
Modifier Datentyp attributname;
```

Das erste zu erstellende Attribut soll öffentlich sein, vom Typ *String* sein und den Attributnamen *Name* tragen. Anhand dieser Schlüsselwörter kann das Attribut wie folgt angelegt werden:

```
public String Name;
```

Die Attribute *Lebenspunkte* und *Erfahrungspunkte* sollen nach außen geschützt, aber vererbbar und ganzzahlig sein. Um ein Attribut nach außen zu schützen und vererbbar zu machen, wird der Modifier *protected* verwendet. Ganzzahlige Werte werden im Datentyp *int* gespeichert.

```
protected int Lebenspunkte;
protected int Erfahrungspunkte;
```

Da ein Charakter bei der Erstellung noch keine Erfahrungspunkte besitzen soll, muss das Attribut *Erfahrungspunkte* bei der Erstellung mit dem Wert 0 initialisiert werden. Dies geschieht im Konstruktor. Der Konstruktor benötigt in diesem Fall keine Parameter. Mit dem Schlüsselwort *this* wird auf das Attribut *Erfahrungspunkte* der Klasse *Charakter* zugegriffen.

```
public Charakter() {
    this.Erfahrungspunkte = 0;
}
```

Damit ein Charakter Erfahrungspunkte erhalten kann wird nun eine öffentliche, nicht abstrakte Methode *erhalteErfahrung* hinzugefügt, mit der die *Erfahrungspunkte* um einen übergebenen, ganzzahligen Wert erhöht werden können. Die Methode muss öffentlich sein, damit man außerhalb der Klasse *Charakter* auf sie zugreifen kann. Es handelt sich um eine nicht abstrakte Methode, da die Berechnung der Erfahrungspunkte für alle Charaktere und erbenden Klassen auf die gleiche Weise funktioniert. Die Methode soll die *Erfahrungspunkte* um einen übergebenen, ganzzahligen Wert erhöhen. Das Schlüsselwort "übergeben" deutet auf einen benötigten Parameter hin. Wie in der vorhergehenden Teilaufgabe soll der Wert wieder ganzzahlig, also vom Typ *int* sein. Da die Methode nichts zurückgeben soll, erhält sie den Rückgabetyp *void*. Da der Parameter der Methode das Attribut der Klasse überdeckt, muss die Zuweisung mit *this* erfolgen.

```
public void erhalteErfahrung(int Erfahrungspunkte) {
    this.Erfahrungspunkte = this.Erfahrungspunkte + Erfahrungspunkte;

    // Alternative
    // this.Erfahrungspunkte += Erfahrungspunkte;

}
```

In der nächsten Teilaufgabe soll eine Methode erstellt werden, die die aktuelle Stufe des Charakters berechnet. Diese Methode soll genau wie *erhalteErfahrung* öffentlich und nicht abstrakt sein. Dadurch wird die Stufe für alle Arten von Charakteren gleich berechnet. Die Methode benötigt keinen Parameter, soll aber einen ganzzahligen Wert zurückgeben. Da die Methode einen Wert zurückgeben soll, benötigt sie den Rückgabetyp *int*. Die Formel zur Berechnung der Stufe ist in der Aufgabe gegeben. Es wird eine neue Variable vom Typ *int* erstellt. In dieser Variable wird das Ergebnis der Berechnung gespeichert. Anschließend wird die Variable zurückgegeben.

```
public int gibStufe() {
    int Stufe = 1 + Erfahrungspunkte / 100;
    return Stufe;
}
```

In der letzten Teilaufgabe wird eine nicht abstrakte Methode, öffentliche Methode *verliereLebenspunkte* erstellt, die den Verlust von Lebenspunkten und eventuellen Tod eines Charakters modellieren soll. Diese Methode soll wieder einen ganzzahligen Parameter erhalten, dessen Wert von den aktuellen Lebenspunkten abgezogen wird. Anschließend muss überprüft werden, ob die Lebenspunkte auf null oder weniger gesunken sind. Ist das der Fall, soll eine vorgegebene Nachricht auf dem Bildschirm ausgegeben werden. An dieser Aufgabe wird der Unterschied zwischen "zurückgeben" und "(auf dem Bildschirm) ausgeben" deutlich. Falls eine Methode einen Wert zurückgeben soll, benötigt sie einen Rückgabetyp und ein *return*. Wenn nichts zurückgegeben werden soll, ist der Rückgabetyp *void*. Falls etwas auf dem Bildschirm ausgegeben werden soll, wird *System.out.println("Text")* verwendet.

```
public void verliereLebenspunkte(int schaden) {
    this.Lebenspunkte = this.Lebenspunkte - schaden;
    // Alternative:
    // this.Lebenspunkte -= schaden;

    if (this.Lebenspunkte <= 0) {
        System.out.println(this.Name + "ist gestorben");
    }
}
```

Die Klasse *Charakter* sieht am Ende folgendermaßen aus:

```
abstract class Charakter {
    // Attribute
    public String Name;
    protected int Lebenspunkte;
    protected int Erfahrungspunkte;

    // Konstruktor
    public Charakter() {
        this.Erfahrungspunkte = 0;
    }

    public void erhalteErfahrung(int Erfahrungspunkte) {
        this.Erfahrungspunkte = Erfahrungspunkte;
    }

    public int gibStufe() {
        int Stufe = 1 + Erfahrungspunkte / 100;
        return Stufe;
    }

    public void verliereLebenspunkte(int schaden) {
        this.Lebenspunkte = this.Lebenspunkte - schaden;
        // Alternative:
        // this.Lebenspunkte -= schaden;

        if (this.Lebenspunkte <= 0) {
            System.out.println(this.Name + "ist gestorben");
        }
    }
}
```

2 Zauberer und Zwerge, Aufgabenteil 2

Die Spieler dürfen sich bei der Wahl ihres Charakters zwischen Zauberern und Zwergen entscheiden. Um diese Verhalten zu modellieren, schreiben Sie zwei neue Klassen *Zauberer* und *Zwerg*, die jeweils von der Klasse *Charakter* abgeleitet werden.

1. Die Klasse *Zauberer* erhält zusätzlich ein privates, ganzzahliges Attribut *Zauberpunkte*.
2. Die Klasse *Zwerg* erhält ein zusätzliches privates, ganzzahliges Attribut *Ruestungspunkte*.
3. Schreiben Sie ebenfalls einen Konstruktor für die jeweilige neue Klasse, der alle Attribute entsprechend auf die ihm übergebenen Parameter initialisiert. Der Konstruktor der Klasse *Zauberer* soll dabei das Attribut *Lebenspunkte* mit den übergebenen Zauberpunkten multiplizieren.
4. Die Klasse *Zwerg* erhält eine öffentliche Methode *verliereLebenspunkte*, die die entsprechende Methode der Klasse *Charakter* überschreibt. In dieser Methode sollen die Lebenspunkte nicht um den vollen Schadenswert reduziert werden. Es wird zunächst der Wert: $Schaden - Ruestungspunkte$ berechnet. Ist dieser neue Wert größer 0, wird dieser Betrag von den Lebenspunkten abgezogen; ist dieser jedoch kleiner 0, so geschieht nichts. Sie dürfen dafür die Methode *verliereLebenspunkte* der übergeordneten Klasse *Charakter* verwenden.

Zu Beginn der Aufgabe sollen zwei Klassengrundgerüste für die Klassen *Zauberer* und *Zwerg* erstellt werden. Beide Klassen sollen von der zuvor erstellten Klasse *Charakter* abgeleitet werden. Die Ableitung bzw. Vererbung wird durch das Schlüsselwort *extends* gekennzeichnet.

```
// Klasse Zauberer:
class Zauberer extends Charakter {

}


// Klasse Zwerg:
class Zwerg extends Charakter {

}
```

Als nächstes werden den beiden Klassen zusätzliche Attribute hinzugefügt, durch diese sie sich voneinander und von der Oberklasse unterscheiden sollen. Bei beiden Attributen wird die Sichtbarkeit und der Datentyp wieder durch die Wörter "privat" und "ganzzahlig" deutlich. Die Klassen sehen anschließend folgendermaßen aus:

```
// Klasse Zauberer:
class Zauberer extends Charakter {
    private int Zauberpunkte;
}


// Klasse Zwerg:
class Zwerg extends Charakter {
    private int Ruestungspunkte;
}
```

Weiterhin sollen beide Klassen einen Konstruktor erhalten, in dem jeweils alle Attribute einen Wert erhalten. Im Falle des Zauberers werden die erhaltenen Lebenspunkte mit den Zauberpunkten multipliziert. Die erforderlichen Parameter sind *Name*, *Lebenspunkte*, beim Zwerg *Ruestungspunkte* und beim Zauberer *Zauberpunkte*.

```
// Klasse Zauberer:
class Zauberer extends Charakter {
    private int Zauberpunkte;

    public Zauberer(String Name, int Lebenspunkte, int Zauberpunkte) {
        this.Name = Name;
        this.Lebenspunkte = Lebenspunkte * Zauberpunkte;
        this.Zauberpunkte = Zauberpunkte;
    }
}


// Klasse Zwerg:
class Zwerg extends Charakter {
    private int Ruestungspunkte;

    public Zwerg(String Name, int Lebenspunkte, int Ruestungspunkte) {
        this.Name = Name;
        this.Lebenspunkte = Lebenspunkte;
        this.Ruestungspunkte = Ruestungspunkte;
    }
}
```

Als letztes wird die Methode *verliereLebenspunkte* der Klasse *Zwerg* überschrieben. Dadurch wird erreicht, dass der erlittene Schaden beim Zwerg aufgrund von Rüstung anders berechnet wird. Dabei wird überprüft, ob der Zwerg mehr Schaden erleidet, als er Rüstungspunkte hat. Ist das der Fall, dann wird die Methode *verliereLebenspunkte* der Klasse *Charakter* aufgerufen. Der Zugriff auf die Methode *verliereLebenspunkte* der Oberklasse ist mit dem Schlüsselwort *super* möglich.

```
public void verliereLebenspunkte(int schaden) {
    int erlittenerSchaden = schaden - Ruestungspunkte;

    if(erlittenerSchaden > 0) {
        super.verliereLebenspunkte(erlittenerSchaden);
    }
}
```

Wenn alle Aufgaben vollständig bearbeitet wurden, sehen die beiden Klassen folgendermaßen aus:

```
// Klasse Zauberer:
class Zauberer extends Charakter {
    private int Zauberpunkte;

    public Zauberer(String Name, int Lebenspunkte, int Zauberpunkte) {
        this.Name = Name;
        this.Lebenspunkte = Lebenspunkte * Zauberpunkte;
        this.Zauberpunkte = Zauberpunkte;
    }
}
```

```java
// Klasse Zwerg:
class Zwerg extends Charakter {
    private int Ruestungspunkte;

    public Zwerg(String Name, int Lebenspunkte, int Ruestungspunkte) {
        this.Name = Name;
        this.Lebenspunkte = Lebenspunkte;
        this.Ruestungspunkte = Ruestungspunkte;
    }

    public void verliereLebenspunkte(int schaden) {
        int erlittenerSchaden = schaden - Ruestungspunkte;

        if(erlittenerSchaden > 0) {
            super.verliereLebenspunkte(erlittenerSchaden);
        }
    }
}
```