Programmieren II (Java)

2. Praktikum: Grundlagen Objektorientierung

Sommersemester 2022

Christopher Auer, Tobias Lehner

Abgabetermine

Lernziele

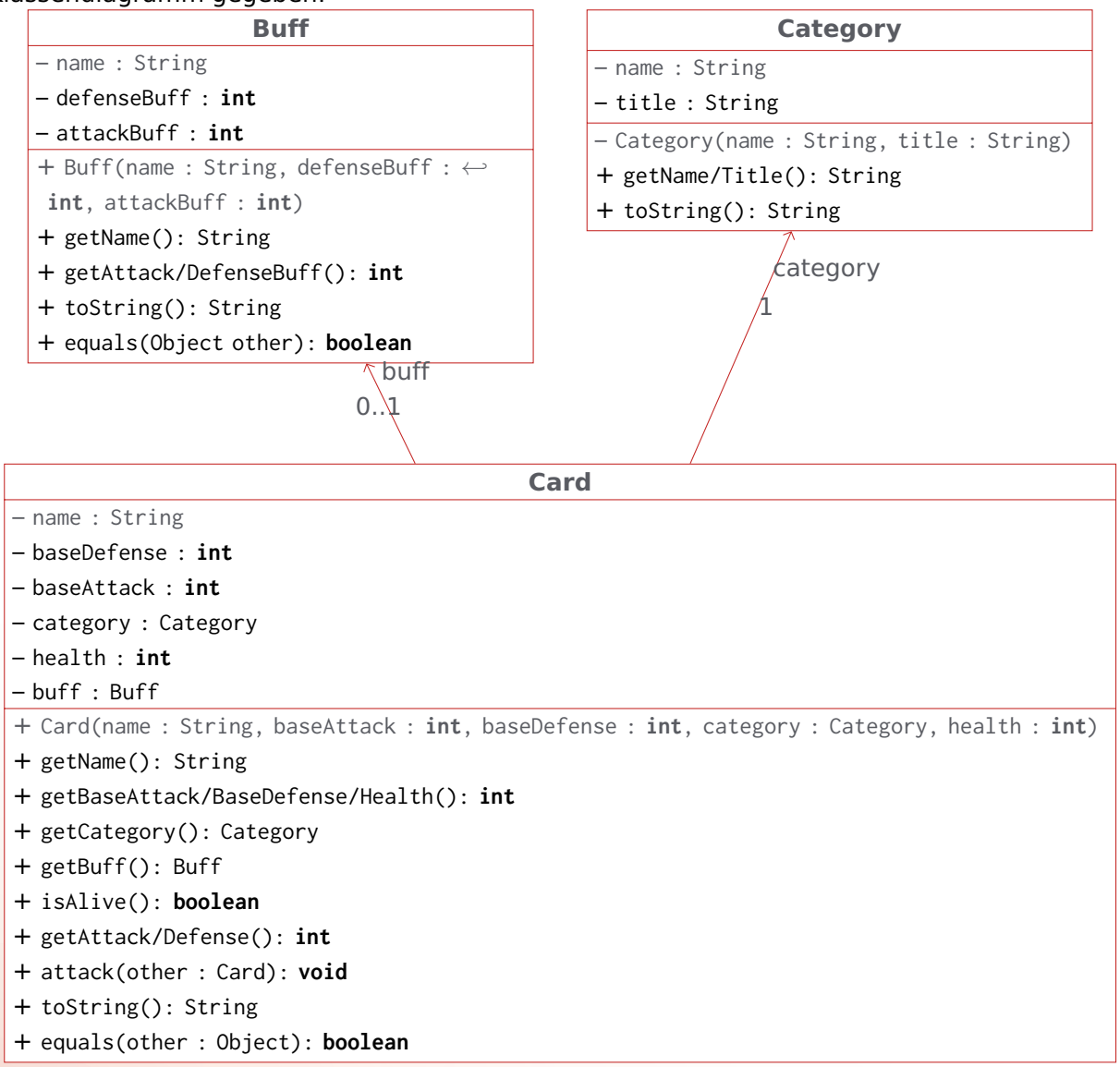
- Implementieren nach einer Spezifikation
- Aufbau von Klassen: Attribute und Methoden
- Konstruktoren
- Getter und Setter
- Java-Standard-Methoden: `equals`, `hashCode`, `toString`
- Dokumentation: `javadoc`
- Testen mit JUnit
- **`enum`**s: definieren, erweitern und verwenden

Hinweise

- Sie dürfen die Aufgaben *alleine* oder zu *zweit* bearbeiten und abgeben
- Sie müssen *4 der 5* Praktika bestehen
- *Kommentieren* Sie Ihren Code
  - Jede *Methode* (wenn nicht vorgegeben)
  - *Wichtige* Anweisungen/Code-Blöcke
  - *Nicht kommentierter* Code führt zu *Nichtbestehen*
- Bestehen Sie eine Abgabe *nicht* haben Sie einen *zweiten Versuch*, in dem Sie Ihre Abgabe *verbessern müssen*
- *Wichtig*
  - Sie sind einer *Praktikumsgruppe* zugewiesen, *nur* in dieser werden Ihre Abgaben *akzeptiert*
  - Beachten Sie dazu die Hinweise auf der Moodle-Kursseite

Aufgabe: Trading Card Spiel

In einem „*Trading Card Game* “ oder „*Deckbuilder Game* “ lässt man Karten gegeneinander antreten. Beispiele dafür sind „Magic: The Gathering“, „Hearthstone“ oder das großartige „Inscryption“. In dieser Aufgabe modellieren wir Karten aus so einem Spiel. Dazu ist folgendes Klassendiagramm gegeben:



- Die Klasse `Card` modelliert ein Karte mit einem *Namen*, einem *Basisangriffswert*, einem *Basisverteidigungswert*, einem *Gesundheitswert* und einer *Kategorie*.
- Die Klasse `Buff` modelliert eine *Kartenmodifikation*, die die Basisangriffs-/Basisverteidigungswerte *modifiziert*.
- `Category` ist ein **`enum`**, das die *Kategorien* der Karten definiert.

Hinweise:

- Erstellen Sie ein Java-Hauptprogramm in einer Klasse `TradingCardsMain`! Die `main`-Methode ist zunächst leer und wird im Laufe der Übung erweitert. Natürlich dürfen Sie die `main`-Methode verwenden um Ihre Implementierung zu *testen*.

- Dokumentieren Sie Ihre Quellcode mit *JavaDoc*!
- Verwenden Sie zum Prüfen auf *Gleichheit* von `Strings` und anderen Objekten die Methode `equals`!
- Prüfen Sie die Parameter jeder Methode auf *Gültigkeit*! Sollte ein Parameter einen ungültigen Wert haben, erzeugen Sie eine `IllegalArgumentException` wie folgt:

```
throw new IllegalArgumentException("Aussagekräftige (!) Fehlermeldung");
```

  Definieren Sie eine *aussagekräftige Fehlermeldung*!
- Testen Sie Ihre Klassen mit den mitgelieferten *JUnit*-Tests! Auf der Moodle-Kursseite finden Sie ein Video, das erklärt wie Sie die Tests in Ihr Projekt integrieren.
- *Tipp*: Kommentieren Sie zunächst alle Methoden in den JUnit-Tests aus. Haben Sie die Implementierung einer Methode abgeschlossen, kommentieren Sie die Tests mit entsprechenden Namen wieder ein. Diese sind nach dem Schema `testMethodenName...` benannt.
- Solange ein Test scheitert, ist *Ihre Implementierung nicht korrekt*. Betrachten Sie in diesem Fall die Fehlermeldung und den Quellcode des gescheiterten Tests.
- *Verändern Sie nicht die Inhalte der JUnit-Tests* um das Problem zu „lösen“!

Die Klasse `Buff`

Die Klasse `Buff` besitzt drei *unveränderliche* Attribute:

- `name` — Name des Buffs, z.B., `"Defense"`, darf *nicht* **`null`** *oder leer (`String.isEmpty`)* sein.
- `defenseBuff` — Veränderung des *Verteidigungswertes* einer Karte. Darf ein beliebiger **`int`**-Wert sein, der zum *Basisverteidigungswert* einer Karte *hinzugezählt* wird (bzw. abgezogen bei einem negativen Wert).
- `attackBuff` — Veränderung des *Angriffswertes* einer Karte. Darf ein beliebiger **`int`**-Wert sein, der zum *Basisangriffswert* einer Karte *hinzugezählt* wird (bzw. abgezogen bei einem negativen Wert).

Hinweis: Die *Anwendung der Veränderungen* auf die Basiswerte implementieren wir *später* in der Klasse `Card`.

- *Deklarieren* Sie Klasse `Buff` mit den oben beschriebenen Attributen!
- *Implementieren* Sie den Konstruktor, der die Parameter *prüft* und die Attribute *initialisiert*! Ein Buff, bei dem `defenseBuff` *und* `attackBuff` *kleiner oder gleich 0* sind, macht keinen Sinn. Sollte das der Fall sein, generieren Sie eine `IllegalArgumentException`.
- *Implementieren* Sie die Getter! Warum macht es hier keinen Sinn *Setter* zu implementieren?
- *Implementieren* Sie die Methode `toString`, die eine `String`-Repräsentation einer Instanz nach folgendem Muster zurückgibt:

```
Defense (D:2, A:0)
```

  für einen Buff mit Namen `"Defense"`, `defenseBuff=2` und `attackBuff=0`.
- *Implementieren* Sie die Methode `equals` wie in der Vorlesung besprochen! Zwei Buffs sind *gleich*, wenn alle Ihre *Attribute* gleich sind.
- *Erweitern* Sie Ihre `main`-Methode indem Sie zwei *lokale Variablen* vom Typ `Buff` deklarieren und wie folgt belegen:

Variable	name	defenseBuff	attackBuff
defenseBuff	Defense	2	0
rageBuff	Rage	-2	4

- Geben Sie die beiden Instanzen über `System.out.println` aus. Welche Methode von `Buff` wird dabei aufgerufen?

Die Enumeration `Category`

Jede Karte ist einer *Kategorie* zugeordnet. Für die Kategorien definieren wir eine *Enumeration* mit zwei Attributen:

- `name` — *Name* der Kategorie (für Ausgaben)
- `description` — *Beschreibung* der Kategorie

Die Werte sind dabei:

enum-Name	title	description
ANIMAL	"Animal"	"Animals roaming the land"
MACHINE	"Robot"	"Robots from the far future"
MAGICAL_CREATURE	"Magical Creature"	"Magical creature from the mystical plane"
PLANT	"Plant"	"Awoken plant that defend themselves"

- *Deklarieren und implementieren* Sie `Category` mit den Attributen, dem Konstruktor und Gettern!
- Erweitern Sie die `main`-Methode, indem Sie die Werte von `Category` auf der Konsole *ausgeben*. Verwenden Sie eine Schleife und die Methode `Enum.values()`.

Die Klasse `Card`

Jetzt können wir die Klasse `Card` mit folgenden Attributes implementieren:

- `name` — *Name* der Karte (*unveränderlich*), z.B., `Wolf`, *nicht* **`null`** *oder leer*
- `baseAttack` — *Basisangriffswert* (*unveränderlich*), muss *mindestens 0* sein
- `baseDefense` — *Basisverteidigungswert* (*unveränderlich*), muss *mindestens 0* sein
- `health` — *Gesundheitswert* (*veränderlich*), ist immer *mindestens 0*. Die Karte ist *„am Leben“*, solange der Wert *größer als 0* ist
- `buff` — angewendeter *Buff*, darf *null* sein

Implementieren Sie `Card` in folgenden Schritten:

- *Deklarieren* Sie `Card` mit den oben genannten Attributen!
- *Implementieren* Sie den Konstruktor, der die Parameter, nach einer *Prüfung*, *initialisiert*. Das Attribut `buff` wird immer mit **`null`** initialisiert.
- Implementieren Sie folgende *Getter* und den *Setter* `setBuff`
  - `getName(): String`
  - `getBaseAttack/BaseDefense/Health(): int`
  - `getCategory(): Category`
  - `getBuff(): Buff`

  Die *Getter* `getAttack` und `getDefense` implementieren wir *später*.
- *Implementieren* Sie die Methode `isAlive()`, die **`true`** liefert wenn `health > 0`, sonst **`false`**!
- Die Methoden `getAttack` und `getDefense` liefern die *effektiven Angriffs- und Verteidigungswerte* nach Anwendung des Buffs (wenn vorhanden). *Implementieren* Sie die beiden Methoden: Sollte `buff` **`null`** sein, geben Sie `baseAttack/Defense` *unverändert* zurück. Sollte `buff` nicht **`null`**

sein, so wird `buff.getAttackBuff()/.getDefenseBuff()` auf `baseAttack/baseDefense` hinzuaddiert.[1] Sollte nach der Addition ein *negativer Wert* herauskommen, geben Sie *0* zurück.

- *Implementieren* die Methode `equals` wie in der Vorlesung besprochen! Zwei `Card`-Instanzen sind *gleich*, wenn alle Attribute *gleich* sind.
- *Implementieren* Sie die Methode `toString`, die die Werte der Attribute in einem `String` darstellt!
- *Zuletzt* implementieren wir die Methode `attack(Card otherCard)` wie folgt:
  - `otherCard` darf *nicht* `null` sein.
  - Beide Karten, `otherCard` und **`this`**, müssen „am Leben“ sein. Sollte das nicht der Fall sein, generieren Sie eine `IllegalArgumentException`. [2]
  - Vergleichen Sie `otherCard.getDefense()` mit **`this`**`.getAttack()`. Sollte der Angriffswert (z.B. 5) *echt größer sein* als der *Verteidigungswert* (z.B., 3), wird die *Differenz* der beiden Werte (2) von `otherCard.health` abgezogen. Bei *Gleichstand* wird *1* abgezogen. Bei einem *größeren Verteidigungswert* passiert nichts. Sollte der `health`-Wert *negativ werden*, setzen Sie ihn auf *0*.

`main`-Methode vervollständigen ★ bis ★★

Vervollständigen Sie die `main`-Methode wie folgt:

- *Deklarieren* Sie vier Karten:

Name	name	baseDef.	baseAtt.	category	initialHealth
wolf	"Wolf"	1	3	ANIMAL	2
lawnMower	"Lawn Mower"	3	1	MACHINE	4
unicorn	"Unicorn"	3	3	MAGICAL_CREATURE	1
manchineelTree	"Manchineel Tree"	1	4	PLANT	5

- Geben Sie im Folgenden die Karten auf der *Konsole* aus, um zu prüfen, ob die Operationen *richtig ausgeführt* wurden.
- *Implementieren* Sie folgende Schritte:
  - `wolf` attackiert `lawnMower`
  - `lawnMower` bekommt `defenseBuff`
  - `wolf` attackiert `lawnMower`
  - `lawnMower` attackiert `wolf`
  - `unicorn` attackiert `manchineelTree`
  - `manchineelTree` bekommt `rageBuff`
  - `manchineelTree` attackiert `unicorn`
- Lassen Sie `unicorn` *nochmals* `manchineelTree` *attackieren*! Was *passiert* und warum?

[1] Bzw. *subtrahiert*, wenn der Buff-Wert *negativ* ist.
[2] Für *Experten*: Eigentlich müsste man hier eine `IllegalStateException` erzeugen