Willkommen zum Tutorium Programmiertechnik II

Stefan Pietzner

05.05.2022

SS 22

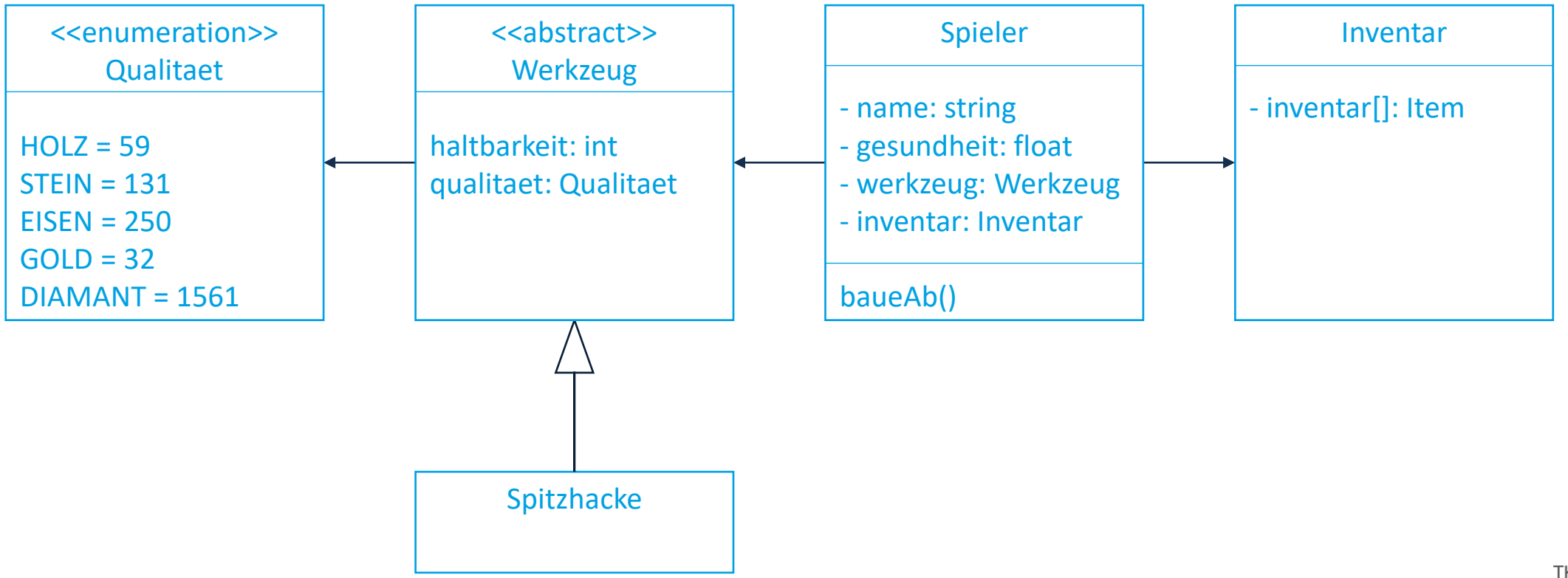
TH Aschaffenburg
university of applied sciences

Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)

Du entwickelst ein kleines Spiel, bei dem es darum geht, Dinge abzubauen und daraus andere Dinge herzustellen (nennen wir es „GrubenCraft“ – Ähnlichkeiten mit einem bereits existierenden Spiel sind rein zufällig :P).

Der Spieler kann zu einem Zeitpunkt ein Werkzeug ausgerüstet haben. Zusätzlich hierzu hat er ein Inventar, in dem die abgebaute Ressourcen verbleiben.

Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)



TH Aschaffenburg
university of applied sciences

Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)

```
public class Spieler implements Cloneable {

String name;
Werkzeug werkzeug = new Spitzhacke(Qualitaet.EISEN);
float gesundheit = 100f;
Inventar inventar = new Inventar();

Spieler(String name) {
                this.name = name;
}

public void baueAb() {

        if (werkzeug == null) {
                System.out.println(name + " hat kein Werkzeug in der Hand!");
                return;
                }

                System.out.println(name + " baut ab: Plock!");
                werkzeug.haltbarkeit--;

                if (werkzeug.haltbarkeit <= 0) {
                                werkzeug = null;
                }
}

protected Spieler clone() {
                try {
                return (Spieler) super.clone();
                }
        catch(CloneNotSupportedException e) {
        e.printStackTrace();
        return null;
        }
}
// hier noch ein paar Getter und Setter, die nicht reingepasst haben
} // End Class
```

Die Spieler-Klasse im Detail

Implementiert das Cloneable-Interface

Unser clone() ruft die Methode Object.clone() auf und castet das Objekt als Spieler-Objekt.
Das ist erst mal in Ordnung, wir können clone()sich aber auch ganz anders verhalten lassen!

Nicht vergessen:
clone() wirft eine CloneNotSupported-Exception, die mit einem try-catch-Block abgefangen werden muss!

TH Aschaffenburg
university of applied sciences

Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)

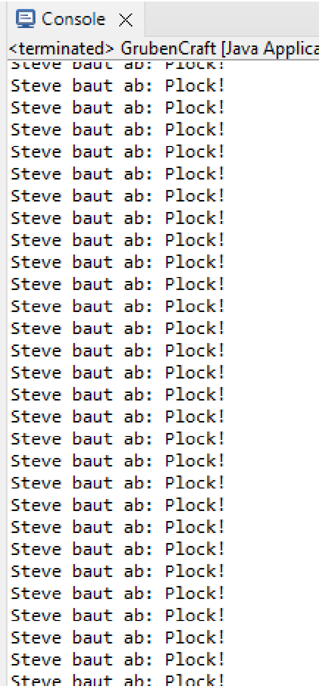
Wir erstellen nun in unserer main-Methode einen Spieler. Diese baut erstmal 245 Blöcke ab, um zu sehen, ob das auch alles klappt:

```
public class GrubenCraft {

public static void main(String[] args) {

Spieler spieler1 = new Spieler("Steve");

for (int i = 0; i < 245; i++) {
        spieler1.baueAb();
}
} //End main
} //End class
```



Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)

Sehr schön!

Jetzt möchtest du einen Multiplayer-Modus implementieren. Also brauchst du mehr als einen Spieler. Das einfachste, was du machst, ist das bestehende Spieler-Objekt zu klonen.

Mithilfe des Cloneable-Interfaces geht das ziemlich leicht…(oder etwa nicht?!)

```
public class GrubenCraft {

public static void main(String[] args) {

Spieler spieler1 = new Spieler("Steve");

for (int i = 0; i < 245; i++) {
              spieler1.baueAb();
              }

Spieler spieler2 = spieler1.clone();
spieler2.setName("Alex");

} //End main
} //End class
```

TH Aschaffenburg
university of applied sciences

Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)

Lassen wir nun Alex auch ein paar Blöcke abbauen, nachdem Spieler Steve seine Blöcke abgebaut hat:

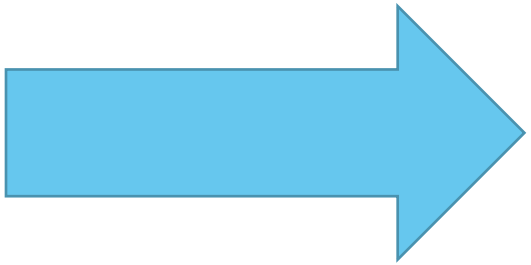
```
public class GrubenCraft {

public static void main(String[] args) {
Spieler spieler1 = new Spieler("Steve");

for (int i = 0; i < 245; i++) {
spieler1.baueAb();
}

Spieler spieler2 = spieler1.clone();
spieler2.setName("Alex");

for (int i = 0; i < 10; i++) {
spieler2.baueAb();
}
} // End main
} // End class
```



Hoppla!

```
Steve baut ab: Plock!
Steve baut ab: Plock!
Steve baut ab: Plock!
Steve baut ab: Plock!
Steve baut ab: Plock!
Steve baut ab: Plock!
Steve baut ab: Plock!
Steve baut ab: Plock!
Steve baut ab: Plock!
Steve baut ab: Plock!
Steve baut ab: Plock!
Alex baut ab: Plock!
Alex baut ab: Plock!
Alex baut ab: Plock!
Alex baut ab: Plock!
Alex baut ab: Plock!
Alex hat kein Werkzeug in der Hand!
Alex hat kein Werkzeug in der Hand!
Alex hat kein Werkzeug in der Hand!
Alex hat kein Werkzeug in der Hand!
Alex hat kein Werkzeug in der Hand!
```

TH Aschaffenburg
university of applied sciences

Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)

So wie es aussieht, ist Alex‘ Spitzhacke bereits nach wenigen Malen kaputt gegangen. Aber sollte sie nicht mindestens so lange halten wie Steves Spitzhacke?

Was ist passiert?

Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)

Ein Objekt wie etwa unsere Spitzhacke ist eine Referenzvariable. Im Gegensatz zu primitiven Datentypen wie Integer, Float, Boolean etc. werden aus Klassen instanziierte Objekte IMMER als Referenzen übergeben, d.h. unser Programm weist sie über ihre Speicheradresse zu.

Sieht so aus, als hätten wir statt einer neuen Spitzhacke unserem neuen Spieler die alte Spitzhacke von Spieler 1 gegeben anstatt sie auch zu klonen…

Wir können das testen, indem wir einige Werte in die Konsole ausgeben lassen.

Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)

Die Lösung:

Wenn wir unseren Spieler klonen, dann müssen wir dafür sorgen, dass auch die Attribute mitgeklont werden.

Hierfür sollte unsere Spitzhacke ebenfalls das Cloneable-Interface implementieren.

Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)

Merke!

- Objekte zu klonen ist eine feine Sache, und Interfaces machen uns hier das Leben leichter.
- Beim Umgang mit Referenzvariablen müssen wir aber immer im Hinterkopf behalten, dass wir versehentlich statt einer *Deep Copy* eine *Shallow Copy* erzeugen können.
- Überall dort, wo Assoziationen auftreten, muss der Programmierer dafür sorgen, dass auch die Assoziationen korrekt mitkopiert werden.
- So lästig das jetzt auch klingt: in manchen Fällen ***wollen*** wir sogar, dass alle Objekte derselben Klasse genau ein Objekt referenzieren...

Aufgabe 5 – GrubenCraft (Cloneable-Interface, Deep & Shallow Copies)

Zum Weitermachen und Nachdenken:

- Fallen dir Beispiele ein, in denen mehrere Objekte auf dasselbe Objekt referenzieren?
- Die Spieler-Klasse hat ein Werkzeug-Attribut – in unserem Beispiel weisen wir der Variablen ein Spitzhacke-Objekt zu! Welches Konzept der objektorientierten Programmierung steckt dahinter?

  → Warum können wir dem Attribut „werkzeug“ in der Spieler-Klasse kein Werkzeug-Objekt zuweisen?
- Lade dir den Code von GitHub herunter und modifiziere ihn! → https://github.com/Daiman15/GrubenCraft

  → Die Inventar-Klasse ist bisher nur ein „Stub“. Kannst du ein funktionierendes Inventar programmieren, das der Spieler mit Items befüllen kann?

  → Wenn der Spieler sich verletzt hat, sollte er etwas Essbares aus seinem Inventar nehmen können und es „essen()“, um seine Gesundheit wieder aufzufüllen.

Danke!

Noch Fragen?

Beantworten wir gerne.

TH Aschaffenburg
university of applied sciences