
CAMP MINECRAFT MODS

– Tag 3 –

Kapitel 1 – Bauwettbewerb

Willkommen zurück an Tag 3 unseres tollen Camps **Minecraft Mods mit den ForscherFreunden**, ich hoffe sehr, dass es euch super geht und ihr viel Spaß habt!

Heute wollen wir zuerst uns mal eure Baukünste anschauen – ich bin mir sicher, dass ihr alle besser als ich bauen könnt!

Danach wollen wir uns heute einmal mit Werkzeugen und Waffen beschäftigen, denn ohne eigene Werkzeuge wäre es doch langweilig eure eigenen Blöcke abzubauen!

Dafür schauen wir uns zunächst aber einen wichtigen Teil der Werkzeuge an – nämlich das, woraus die Werkzeuge und Waffen bestehen werden – euer eigenes Material. Hoffentlich habt ihr alle ein cooles Item an Tag 1 entworfen, welches heute als Grundlage dienen kann.

Kapitel 1.1 – Woraus besteht denn meine Spitzhacke?

Um unser Werkzeug aus einem bestimmten Material stammen zu lassen, könnten wir einfach auf die bereits vorhandenen Materialien zurückgreifen – aber dann bräuchten wir ja keine eigenen Werkzeuge zu erfinden, immerhin gibt es bereits Diamantspitzhacken, -schwerter, -äxte, usw. ...

Stattdessen wollen wir nun ein eigenes Material als Grundlage nutzen – das Item vielleicht welches ihr an Tag 1 erstellt habt? Dafür gehen wir in IntelliJ einmal auf die Klasse *RubinMaterial*, direkt über der *TestMod* Klasse. Hier findet ihr einmal eine Materialklasse, nämlich die von den Rubinen. Diese könnt ihr als Vorlage für euer eigenes Material verwenden, wir wollen die wichtigen Punkte aber trotzdem einmal durchgehen.

Jede Materialklasse setzt sich aus unterschiedlichen Parametern zusammen, die für das Material relevant sind. Da hätten wir für unser Ausrüstungsmaterial einmal:

- Haltbarkeit
- Abbaugeschwindigkeit
- Angriffsschaden
- Abbaulevel
- Verzauberkeit
- Reparaturzutat

Diese Attribute sind in der Klasse allesamt als Funktionen angegeben, nämlich mit einer sogenannten *getter* – Funktion (engl.: *bringer* – Funktion). Diese Funktionen dienen allein dazu, einen Bestimmten Wert dieses Attributes zu erzeugen und zu überliefern. So können wir also ganz einfach diese Werte

für unser Material anpassen. Dafür müssen wir nur die jeweiligen Zahlen in dem *return* Statement anpassen.

```
7
8  /*
9   * Diese Klasse ist die Materialklasse der Vorschauitems "Rubin".
10  * Hier seht ihr einmal alle wichtigen Werte implementiert, um eure eigene Klasse zu erstellen.
11  * Lest dafür Kapitel [2] in der Doku [Tag 3] genau durch.
12  */
13
14
15  5 usages  🔍 ForscherViking
16  public class RubinMaterial extends AusruestungsMaterial {
17
18      1 usage  🔍 ForscherViking
19      @Override
20      public int gebeHaltbarkeit() { return 3500; }
21
22
23      1 usage  🔍 ForscherViking
24      @Override
25      public float gebeAbbauGeschwindigkeitMultiplikator() { return 11.0f; }
26
27
28      1 usage  🔍 ForscherViking
29      @Override
30      public float gebeAngriffsSchaden() { return 0; }
31
32
33      1 usage  🔍 ForscherViking
34      @Override
35      public int gebeAbbauLevel() { return 5; }
36
37
38      1 usage  🔍 ForscherViking
39      @Override
40      public int gebeVerzaubarkeit() { return 15; }
41
42
43      1 usage  🔍 ForscherViking
44      @Override
45      public Item[] gebeReparaturZutat() {
46          return new Item[]{
47              Mod.HoleItem( name: "ruby"),
48              Items.DIAMOND
49          };
50      }
51  }
```

Aber was machen die einzelnen Attribute denn? Die meisten davon sollten recht selbsterklärend sein, aber gehen wir die einmal durch.

Haltbarkeit gibt die Haltbarkeit als ganze Zahl an, also wie viele Ticks beim Abbauen das Werkzeug aushält. Spielt damit ruhig etwas herum, aber irgendwo zwischen 1000 und 4000 ist ein guter Start.

Abbaugeschwindigkeit – wie der Name verrät – gibt die Geschwindigkeit zum Abbauen. Hier einen Wert als Float-Wert zwischen 9.0f und 15.0f angeben, sollte ganz gut sein.

Angriffsschaden ist eine Sache für sich... ihr könnt hier zwar einen Wert angeben, jedoch werden wir später beim Erstellen der Werkzeuge und Waffen selbst die Werte für den Angriffsschaden und -geschwindigkeit setzen. Hier könnt ihr den Wert also auch auf 0 lassen.

Abbaulevel gibt an, welche Blöcke abgebaut werden können. erinnert ihr euch an die Blöcke von gestern? Die haben wir in Level eingetragen... Level 3 war Diamant, Level 4 Netherite und wir hatten gesagt, dass Level 5 und höher nur mit custom Werkzeugen abbaubar wäre. Das haben wir jetzt hier als Möglichkeit: Gebt ihr hier Level 5 oder höher an, kann dieses Werkzeug alle Blöcke in dieser Kategorie und niedriger abbauen.

Verzauberkeit ist ein etwas seltsamer Wert. Er gibt an, wie wahrscheinlich es ist, dass an einem Verzauberungstisch bessere Verzauberung angezeigt werden.

Und die Reparaturzutaten sind als Liste angegeben, hier schreibt ihr also die Items rein, mit denen am Amboss Werkzeuge dieses Materials wieder repariert werden können sollen.

Wunderbar!

Wenn wir nun unser eigenes Material erstellen wollen – wie machen wir das?

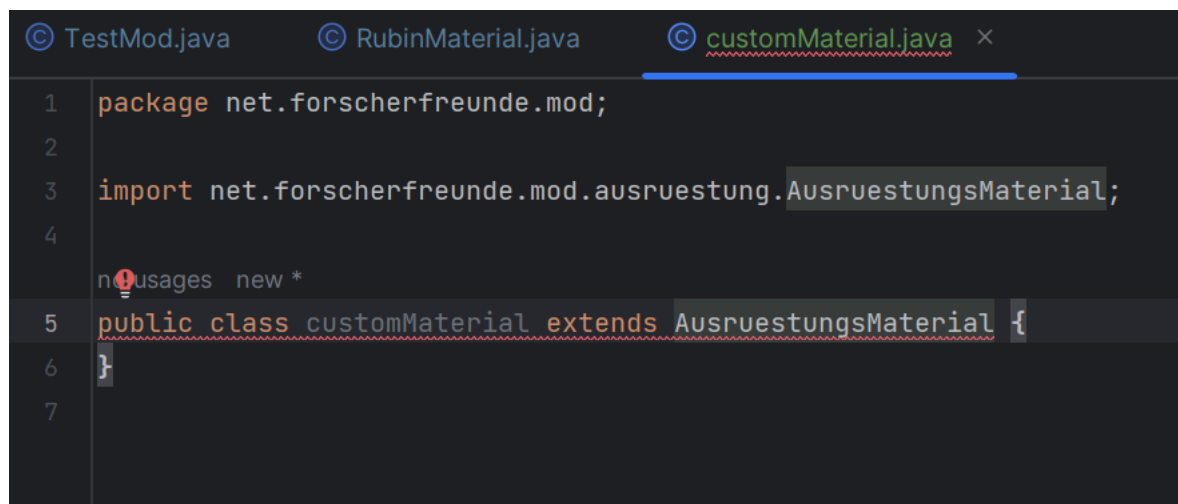
Kapitel 1.2 – unser eigenes Material

Um ein eigenes Material zu erstellen, suchen wir den Ordner *mod* in der Ordnerstruktur (*main* -> *java* -> *net* -> *forscherfreunde* -> *mod*), machen dort Rechtsklick und erstellen eine neue Klasse, welche wir dann *CustomMaterial* nennen, wobei ihr dann *custom* natürlich durch den Namen eures Materials ersetzt. Schreibt gerne Material groß, so ist es einheitlich mit dem Rest, und einfacher zu verstehen, welche Klasse wofür zuständig ist.

In dieser Klasse schreiben wir jetzt hinter den Namen der Klasse, und noch vor der Klammer auf folgendes hin:

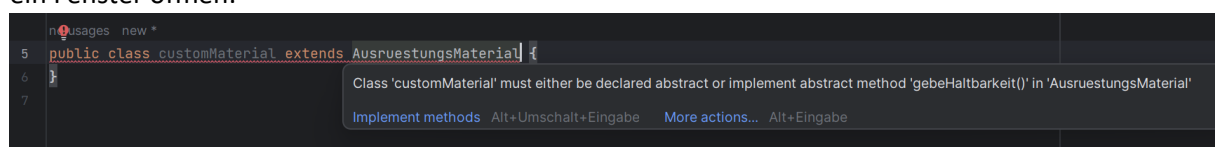
extends AusruestungsMaterial

Das sollte dann in etwa so aussehen:



```
© TestMod.java  © RubinMaterial.java  © customMaterial.java x
1 package net.forscherfreunde.mod;
2
3 import net.forscherfreunde.mod.ausruestung.AusruestungsMaterial;
4
5 public class customMaterial extends AusruestungsMaterial {
6 }
7
```

Jetzt könnt ihr einmal mit der Maus über der rot unterstrichenen Zeile schweben, dann sollte sich ein Fenster öffnen:



```
1 public class customMaterial extends AusruestungsMaterial {
2 }
3
```

Class 'customMaterial' must either be declared abstract or implement abstract method 'gebeHaltbarkeit()' in 'AusruestungsMaterial'

Implement methods Alt+Umschalt+Eingabe More actions... Alt+Eingabe

Dort drücken wir auf *Implement methods*, woraufhin wir in dem neuen Fenster einfach *ok* drücken.

Nun sollten alle Funktionen, die wir zuvor gesehen haben, auch hier sein:

```
1 package net.forscherfreunde.mod;
2
3 import net.forscherfreunde.mod.ausruestung.AusruestungsMaterial;
4 import net.minecraft.item.Item;
5
6 no usages new *
7 public class customMaterial extends AusruestungsMaterial {
8     1usage new *
9     @Override
10     public int gebeHaltbarkeit() {
11         return 0;
12     }
13
14     1usage new *
15     @Override
16     public float gebeAbbauGeschwindigkeitMultiplikator() {
17         return 0;
18     }
19
20     1usage new *
21     @Override
22     public float gebeAngriffsSchaden() {
23         return 0;
24     }
25
26     1usage new *
27     @Override
28     public int gebeAbbauLevel() {
29         return 0;
30     }
31
32     1usage new *
33     @Override
34     public int gebeVerzaubarkeit() {
35         return 0;
36     }
37
38     1usage new *
39     @Override
40     public Item[] gebeReparaturZutat() {
41         return new Item[0];
42     }
43 }
```

Dann passt ihr nur noch die Items an, per Befehl

Mod.HoleItem(„name“)

Könnt ihr auf eure eigenen erstellten Items zugreifen, einfach den Namen des Items angeben (siehe: *Mod.HoleItem(„ruby“)*).

Per Befehl

Items.

Gefolgt von dem Identifier des Items (wenn ihr das schreibt, wird euch direkt eine Liste vorgeschlagen, einfach das item raussuchen), als Englisches großgeschriebenes Wort angeben (siehe: *Items.DIAMOND*).

Dann sollte euer eigenes Ausrüstungsmaterial soweit fertig sein, Glückwunsch!

Also können wir uns jetzt anschauen, wie wir eigene Werkzeuge und Waffen mit diesem Material machen.

Diese können wir nun mit unseren gewünschten Werten füllen.

Also zuerst die Haltbarkeit

Dann die Abbaugeschwindigkeit

Folgend der Angriffsschaden (könnt ihr einfach lassen)

Das Abbaulevel (falls ihr einen Block in Kategorie 5 gepackt habt, dann am besten mindestens 5)

Verzaubarkeit – hier gerne einen Wert zwischen 10 und 20

Und zuletzt ändern wir ganz unten die Reparaturzutaten Liste etwas ab. Dafür kopieren wir einfach die Zeilen zwischen den geschweiften Klammern in der *RubinMaterial* Klasse, und fügen sie hier ein.

Alternativ könnt ihr es einfach dort selbst ändern, so dass es wie folg aussieht:

```
@Override
public Item[] gebeReparaturZutat() {
    return new Item[]{
        Mod.HoleItem( name: "ruby"),
        Items.DIAMOND
    };
}
```

Kapitel 1.3 – Eigene Werkzeuge und Waffen

Der Anfang – wie vielleicht erwartet – findet wieder in Paint.net statt, mit der Textur. Also nehmt euch doch jetzt einmal ein paar Minuten, schöne Texturen für eure Werkzeuge zu erstellen. Macht zu Beginn erst mal nur ein Werkzeug oder Waffe, ihr habt danach noch Zeit weitere zu machen, aber so könnt ihr das Prinzip erst mal verstehen und lernen.

Nehmt euch ein Vorbild an der Textur der Rubin Werkzeuge, diese findet ihr im Ordner *resources* -> *assets* -> *forscherfreundemod* -> *textures* -> *item* als *ruby_pickaxe* usw.

Wenn ihr die Textur fertig habt, speichert sie – genau wie euer Item an Tag 1 – in dem oben genannten Ordner unter *item* ab, mit dem Namen des Materials + *_* gefolgt von der Werkzeug-/Waffen Art.

Also in etwa so:

custom_spitzhacke.png

neue_axt.png

meine_schaufel.png

usw.

Ich denk das sollte klar sein. Natürlich alles KLEIN geschrieben und OHNE LEERZEICHEN. Stattdessen verwenden wir immer diese *_* Symbole.

Nun müssen wir nur noch in der *TestMod* Klasse unter der Kategorie *TestWerkzeuge und Waffen* unser Werkzeug registrieren. Dafür schreiben wir eine der 5 Funktionen:

```
Mod.SpitzhackeHinzufuegen(„name“, new unserMaterial(), [Angriffsschaden als ganze Zahl],  
[Angriffsgeschwindigkeit als Float Wert] );
```

```
Mod.AxtHinzufuegen(„name“, new unserMaterial(), [Angriffsschaden als ganze Zahl],  
[Angriffsgeschwindigkeit als Float Wert] );
```

```
Mod.SchaufelHinzufuegen(„name“, new unserMaterial(), [Angriffsschaden als ganze Zahl],  
[Angriffsgeschwindigkeit als Float Wert] );
```

```
Mod.SchwertHinzufuegen(„name“, new unserMaterial(), [Angriffsschaden als ganze Zahl],  
[Angriffsgeschwindigkeit als Float Wert] );
```

```
Mod.FeldhackeHinzufuegen(„name“, new unserMaterial(), [Angriffsschaden als ganze Zahl],  
[Angriffsgeschwindigkeit als Float Wert] );
```

Und setzen natürlich immer den Namen und die Werte passend ein. Statt „name“, den Namen des Werkzeuges (also der Textur-Name), statt *unserMaterial()* der Name von eurer Material-Klasse, und die beiden eckigen Klammern lasst ihr natürlich weg, und schreibt nur den Wert hin, also entweder eine ganze Zahl für den Angriffsschaden, oder einen float Wert für die -geschwindigkeit (also z.B.: 1.0f).

Bei den Werten könnt ihr euch, wie immer, an den Rubin Vorlagen orientieren.

Kapitel 1.4 – Die Werkzeuge und Waffen – Modelle und Name

Nun brauchen wir, wie zuvor bei den Items auch, nur noch die Modelle, damit es im Spiel auch nach etwas aussieht, und die Sprache.

Für die Modelle gehen wir in die Klasse *datagen* -> *ModModelsProvider*, und schauen uns jetzt die Zeilen von den Werkzeugen an.

```
//Werkzeuge - siehe Kapitel [1.4] an [Tag_3]  
itemModelGenerator.register(ModItems.GetItem( name: "ruby_axe"), Models.HANDHELD);  
itemModelGenerator.register(ModItems.GetItem( name: "ruby_hoe"), Models.HANDHELD);  
itemModelGenerator.register(ModItems.GetItem( name: "ruby_shovel"), Models.HANDHELD);  
itemModelGenerator.register(ModItems.GetItem( name: "ruby_pickaxe"), Models.HANDHELD);  
itemModelGenerator.register(ModItems.GetItem( name: "ruby_sword"), Models.HANDHELD);
```

An Sich sieht das erst mal super ähnlich zu den normalen Items aus – und das ist es auch. Ihr müsst nur den Namen von eurem Werkzeug anpassen beim Neuschreiben der Zeile.

Das, was dahinter steht ist jetzt aber unterschiedlich zu den Items. Anders als bei den Items, haben wir hier stehen:

Models.HANDHELD

Statt:

Models.GENERATED

Das bedeutet, dass die .json Datei jetzt mit einem anderen Wert erzeugt wird, so dass das Werkzeug im Spiel auch tatsächlich in der Hand angezeigt wird, so wie es sollte.

Ihr könnt spaßeshalber mal die Zeile des Items kopieren, und schauen, wie es dann im Spiel aussieht!

Die Sprachdatei kennen wir ja bereits gut, dort schreibt ihr einfach euer Werkzeug, genau wie ein Item hinzu (Die Datei *de_de.json* in *assets* -> *lang*).

Kapitel 1.5 – Rezepte für unsere Werkzeuge und Waffen

Da wir nun unsere großartigen Werkzeuge und Waffen im Spiel haben, wollen wir uns um die Funktionen kümmern, was dieses Kapitel dann abschließen sollte. Die Rezepte.

Dafür könnt ihr euch einmal die Doku über die Rezepte anschauen, solltet ihr das noch nicht gemacht haben. Diese findet ihr in dem Ordner, wo ihr auch diese Doku gefunden habt.

Auf dieser Grundlage erstellen wir nun eine Rezeptdatei in dem Ordner *resources* -> *data* -> *forscherfreundemod* -> *recipes* und nennen dieses dann *unsermaterial_werkzeugtyp.json* und ersetzen natürlich unseren Materialnamen und den Werkzeugtypen. Alles klein, alles ohne Leerzeichen – so wie immer. Die Datei erstellen wir dann nach Vorgabe der Rezepte – Doku, als ein *shaped* Rezept.

Sobald ihr damit zufrieden seid, erstellt ihr am besten auch die Rezepte für alle anderen Werkzeuge und Waffen, die ihr zuvor erstellt habt. Dann könnt ihr einmal ins Spiel gehen, und die Rezepte ausprobieren.