Jimmy Koene

In **Full Color**

Sams Teach Yourself Mod Development for Minecraft®

Second Edition

in 24 Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Jimmy Koene

Sams **Teach Yourself**

Mod Development for Minecraft®

in 24 Hours

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself Mod Development for Minecraft® in 24 Hours

ISBN-13: 978-0-672-33763-5

ISBN-10: 0-672-33763-0

Library of Congress Control Number: 2015913370

Printed in the United States of America

First Printing November 2015

Trademarks

Warning and Disclaimer

Special Sales

Editor-in-Chief
Greg Wiegand

Executive Editor
Rick Kughen

Development Editor
Mark Renfrow

Managing Editor
Sandra Schroeder

Project Editor
Mandie Frank

Copy Editor
Barbara Hacha

Indexer
Lisa Stumpf

Proofreader
Paula Lowell

Technical Editor
Boris Minkin

Editorial Assistant
Kristen Watterson

Designer
Mark Shirar

Compositor
codeMantra

Contents at a Glance

Table of Contents

Part VI: Final Pointers

APPENDICES

About the Author

Jimmy Koene maintains one of the world's most popular Minecraft mod websites (www.wuppy29.com/minecraft/modding-tutorials/forge-modding/). He is one of the only Minecraft modders who consistently maintains and updates his online tutorials. Also an expert Java and C++ programmer and Unity developer, he has written an Android Minecraft app that has been downloaded more than 45,000 times, along with various other smaller apps. He also maintains several Minecraft mods that cover many aspects of Minecraft, ranging from slight changes to how food works to mods that completely change the way you play.

Dedication

To the Minecraft mod community, who make Minecraft one of the best-selling and amazing games there is. Thanks to the community, I started modding, which led to the writing of this book.

Acknowledgments

First of all, I want to thank Rick Kughen, who approached me about writing a book about Minecraft mod development. Before he emailed me with his ideas, I never thought this would be possible. One of the biggest problems with developing mods for a game such as Minecraft is that the code you are working with constantly changes. Every update of Minecraft or Forge, which is the application programming interface (API) used in this book, may change quite a lot of the code. Thankfully, eBooks can be easily updated, and Rick told me about the possibility of having an online database where I could frequently add updates and changes to the book. This made it possible to write this book.

Some other people I should thank are Markus "Notch" Persson, who created Minecraft. Without his idea, I never would have started programming, and this book would never have been written. There are many others who helped create Minecraft and the modding community. Some of those people are Jeb, Dinnerbone, Searge, ProfMobius, R4wk, CodeWarrior, ZeuX, Risugami, cpw, AbrarSyed, and LexManos. These are the creators of MCP, MCEdit, Techne, ModLoader, and Forge. All these things are (indirectly) used in this book. Finally, I would like to thank everybody who worked on the book. This includes the editors, Mark Renfrow and Rick Kughen, the reviewers, and everybody else who did anything related to the book. Without all these people, this book would never have been possible. I especially want to thank LexManos, the creator of Forge. His expertise in this field helped a lot to improve this book.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: consumer@samspublishing.com

Mail: Sams Publishing
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Introduction

Minecraft is a great game. It's one of the best-selling PC games, and it's exceedingly popular. One of the many reasons why it is so popular is because there are mods. If you picked up this book, you likely already have an interest in Minecraft, and you might have already played with some mods. This book teaches you how to create these mods yourself.

Creating mods yourself has two big advantages. First, you can improve the game in the way you want, shaping the mod to exactly fit your needs and making the game more fun for you to play. Second, modding is an easy way to get into programming, which is a booming industry, and, according to many people (including me), modding is a really fun and interesting thing to do.

By the end of this 24-hour book, you will be able to create your own mods and be well on your way to programming with Java. Each hour covers an important aspect of Minecraft mod development, including various items, blocks, and entities. You will learn a lot about each of these subjects by following along with each hour, actively participating in the hands-on Try It Yourself activities, and completing the Workshops and Exercises at the end of each hour. You will also gain more information on how to use your development environment, learn things by yourself, and discover many other topics provided in various Notes and Tips sprinkled throughout the book.

Who Should Read This Book

This book is written for anyone who likes Minecraft and isn't afraid to learn a thing or two about programming and Minecraft.

If you expect a full guide on how to start programming, this book isn't for you. However, if you want to get into programming or already know how to program, this book will certainly help. Creating mods for Minecraft is how I got started with programming, and the same goes for many other mod creators.

It might be helpful to have some familiarity with Java before beginning this book; however, the Java basics are explained as they are used throughout the book. Therefore, it isn't necessary to have prerequisite programming knowledge. It is helpful if you have experience playing Minecraft, though, because many terms used in the game are also used in the book. Additionally, playing Minecraft gives you an insight into how certain aspects of the game work,

which makes it easier to understand what you are creating. When you get to the last hour of this book, it is suggested that you learn Java, but by that time you should already have a reasonable understanding of it, which makes it easier to learn. Following the last hour is an appendix that contains even more code, which you will likely need in your own mod.

How This Book Is Organized

Learning everything Minecraft has to offer in just 24 hours might seem like a lot to learn—and it is. This book doesn't cover the creation of every single thing in Minecraft. Instead, it covers creating the most commonly used parts, which gives you the knowledge to make the more specific things. The following breakdown contains the details of what you will learn in each of the hours:

- In Hour 1, "Setting Up the Minecraft Development Environment," you set up the environment in which you will start developing your Minecraft mods. In this hour, you learn about most of the programs you will be using throughout the book.
- In Hour 2, "Creating the Basics for Forge," you start coding things. In this hour, you create the framework of your mod, which you build on in all the other hours.
- Hour 3, "Working with Recipes and Other Small Modifications," is the first hour where you add custom content to Minecraft in the form of crafting and smelting recipes and more.
- In Hour 4, "Making Your First Item," you create your first item. Starting in this hour, you add the first truly unique and interesting content to your mod. Four hours until you start adding unique content is a long time, but it is required, because you can't mod without the code or a mod class.
- In Hour 5, "Creating Multiple Items in a Smart Way," you expand the code you made in Hour 4 to make sure your mod looks clean and works well with other mods.
- In Hour 6, "Cooking Up a Food Item," your goal is to create an item that the player can eat and that heals the player. This is one of the many custom item types you can create.
- Hour 7, "Making Your Own Tools," is another hour containing a custom item type. Here you learn how to make custom pickaxes, shovels, and completely new tools.
- In Hour 8, "Creating Armor," you learn about another custom item type—armor. You learn how to create armor and then how to customize it.
- In Hour 9, "Making Your First Block," similar to Hour 4, you learn about another important aspect of Minecraft. In this hour, you learn about the things the world is made of—blocks.
- In Hour 10, "Block States" you learn the same thing you learned in Hour 5 about items, but for blocks.

- In Hour 11, "Making Your Blocks Unique," you learn various ways to make your blocks interesting. You learn how to make half blocks, plants, and several other items.
- In Hour 12, "Creating a Tile Entity," you learn about one of the most complicated parts of Minecraft—a tile entity.
- Hour 13, "Generating Ores," is the first hour in which you generate something. In this hour, you learn how to create a class that will do all the world generation for you. It uses premade code to generate your ores.
- In Hour 14, "Generating Plants," you take what you learned in Hour 13 one step further. You now learn how to create custom code to generate things in the world.
- In Hour 15, "Using MCEdit," you learn how to use a program called MCEdit to create a structure that will be generated in your mod in Hour 16.
- In Hour 16, "Generating Your Structure," you make the structure created in the previous hour generate in the world of Minecraft.
- In Hour 17, "Learning About Entities," you learn what entities are, which kind of entities there are, and how to create the fundamentals for an entity.
- In Hour 18, "Creating an Entity Model Using Techne," you learn how to create a model for the entity you will create in Hour 19. This uses a tool called Techne, which is a piece of modeling software created specifically for Minecraft mods.
- In Hour 19, "Coding a Mob," you use the model you created in Hour 18, combine it with code from Hour 17, and then add some more code to create something similar to a zombie, but slightly different.
- In Hour 20, "Creating a Throwable," you learn about a different kind of entity that can be thrown by the player.
- In Hour 21, "Editing Vanilla Minecraft," you learn how to edit Minecraft in such a way that multiple mods can edit the same thing without it breaking the game.
- In Hour 22, "Structuring Your Mod," you learn how to clean up your mod a little. At this point, several files will look quite crowded, which can be harmful to your experience. After this hour, you should be able to continue coding in an environment that is much easier to use.
- Hour 23, "Releasing Your Mod," is a very important hour. In this hour, you learn how to use your mod outside of your development environment and how to distribute it to users.
- In Hour 24, "What's Next," you learn how to continue working with Minecraft mod development after completing this book. This book is more than just a guide on how to create what is covered here. The goal is that you learn how to create things yourself.

Conventions Used

This book uses several conventions to make it easier for you to understand what is being explained.

Try It Yourself

The Try It Yourself activities are one of the most important conventions in this book. The Try It Yourself activities provide hands-on opportunities to actively experience and engage in the topic covered in that hour. These hands-on activities provide an opportunity to learn by doing. This is the best way to learn Minecraft modding.

Notes

The Note elements provide extra tidbits of information. These interesting tidbits provide supplemental content or expand the information given in the nearby text.

Tips

A Tip element provides extra information that can be useful when you are modding. This information can range from tips on how to better use a program to certain things you might want to change in the code. Tips identify handy tricks or expert advice that will help you along the way.

Cautions

The Caution element warns you when an action you might take could have dire consequences later. This cautionary text warns you of potential hazards and provides advance warning of outcomes you want to avoid.

Code Listings

Code listings are another helpful element you'll find in this book. Code listings provide sample snippets of code that relate to the topic at hand. You'll be able to examine the sample code to get a feel for how your code should be written.

In this book, formal code listings are always surrounded by two lines to show where the code starts and where it ends. The text surrounding the listing explains where this code should be placed, if this isn't clear from the code itself.

When a line of code is too long to fit on one line of text, it is wrapped to the next line. In that case, the continuation is preceded with a code-continuation arrow (➥).

Downloading the Code and Resources

Sometimes your mod will just not work, and no matter how much you look at it, you can't find the error. In this case, it might be helpful to copy the code used in this book directly into your workspace. Examining the sample code and comparing it with the code you have written might help you locate and correct the error in your code. Additionally, not everyone is good at making textures, so you might want to obtain the textures used in this book. Both of these things can be accessed by registering your book:

1. Go to www.informit.com/title/9780672337635.
2. Click Register Your Product and log in or create a new account.
3. Enter this ISBN: 9780672337635. This is the ISBN of the print book and must be used to register every edition.
4. Click the Access Bonus Content link in the Registered Products section of your account page, which will take you to the page where your downloadable content is available.

Furthermore, as mentioned in the Acknowledgments, the code used to make mods for Minecraft can change quite often. To make sure this book will remain useful in the future, there are online updates and errata, which can be found at www.informit.com/title/9780672337635 on the Updates tab.

This page intentionally left blank

HOUR 3

Working with Recipes and Other Small Modifications

What You'll Learn in This Hour:

- The different kinds of recipes in Minecraft
- Creating all those recipes
- Changing dungeon spawns
- Changing chests

This is the first hour where you actually write your own code to get started for your own mod. This hour contains some of the most basic Minecraft modifications, which include *recipes*, *dungeon* possibilities, and generated *chests*. Thanks to Forge you can add every kind of recipe in Minecraft. You can also change the mobs that can be spawned in a dungeon. This can be used to add your own mob, but you can also create, for example, the possibility of a creeper dungeon. Finally, Forge also enables you to change the contents of any *vanilla* generated chest. Vanilla is the base of Minecraft without any mods installed.

Learning About Recipes in Minecraft

In Minecraft there are multiple kinds of recipes. There is a difference between smelting iron ore and crafting an iron pickaxe with it. This should be quite logical, because you do both in a different block. However, a difference also exists between crafting something like planks and a pickaxe.

There are three types of crafting recipes:

- Crafting recipes are recipes where you need certain items or blocks in a specified pattern. Another name for these is standard or shaped recipes. An example of a recipe like this is an iron pickaxe.
- Shapeless recipes are recipes without a defined shape. They also require certain items and blocks, but the pattern doesn't matter. An example of a shapeless recipe is a flint and steel.
- Smelting recipes are all used in a furnace. Smelting recipes don't have a shape and never have more than one item or block as an input.

For all the recipe types, you can return only a single item or block, but you can return several of the same item or block.

Crafting a Recipe

Now you will finally start modding the game. Recipes may not be that big of a change, but several combined can change the way the game is played, especially in combination with custom `Items`, which are explained in Hour 4, "Making Your First Item."

Recipe code has to be added in the mod file. Listing 3.1 is the mod file from Hour 2 without the `System.out.println` lines.

LISTING 3.1 Clean Mod File

```
package com.wuppy.samsmod;

import net.minecraft.init.Blocks;
import net.minecraftforge.mods.fml.common.Mod;
import net.minecraftforge.mods.fml.common.Mod.EventHandler;
import net.minecraftforge.mods.fml.common.event.FMLInitializationEvent;

@Mod(modid = SamsMod.MODID, version = SamsMod.VERSION)
public class SamsMod
{
    public static final String MODID = "wuppy29_samsmod";
    public static final String VERSION = "1.0";

    @EventHandler
    public void init(FMLInitializationEvent event)
    {

    }
}
```

The code for recipes can look quite complicated to start with and is really hard to understand without starting off with a code example. In the code that follows, you can see an example of a recipe. When you add this code, make sure you import the new files.

```
GameRegistry.addRecipe(new ItemStack(Items.apple),
       "XXX",
       "XXX",
       "XXX",
       'X', Blocks.leaves
);
```

This recipe will look like Figure 3.1.

FIGURE 3.1
The Apple recipe in game.

CAUTION

Code Location

This code, as well as any of the other code in this hour, has to go into the `init` method or it will not work; it will be full of errors.

Using `GameRegistry`

The first part of this line is `GameRegistry`. `GameRegistry` is a file you will frequently use when coding Minecraft Forge mods. This file contains the registry of every recipe, and also items and blocks.

Because there are so many things you can do with this file, it has many methods that all do different things. To create a basic shaped recipe, you need the `addRecipe` method. This method takes two parameters. The first parameter is an `ItemStack`. An `ItemStack` is another class you will use frequently. It's used to handle items or blocks in stacks. This recipe shows the most basic type of `ItemStack`, but you can add two more numbers as parameters. What those do will be shown in a different recipe. When you just use an `Item` or `Block` as parameter, it will create an `ItemStack` with that object with a stack size of 1 and a metadata of 0. Metadata and how to code it are explained in Hour 5, but examples of metadata are dyes and wool blocks.

The first parameter is the `Item` or `Block` the recipe will return. In this case it will return a single apple.

The second parameter in the `addRecipe` mode is a weird one. If you take a look at the parameter in the `GameRegistry` class, it's written as follows:

```
Object... params
```

The `...` means that it's a parameter that can take any amount of values you want. This parameter can be interpreted as some sort of list. Everything you put after the first parameter of the `addRecipe` method will be added in this list and used in the recipe.

The Shape of the Recipes

The way these `Recipes` are done in Minecraft is by typing the shape of the crafting table in the recipe using three strings. Each of the three strings contains three spaces to put an `Item` or a `Block`. Another example of the crafting table space:

```
"ABC",
"DEF",
"GHI",
```

Each letter represents one of the usable crafting areas in the crafting grid in the crafting table If you want one of the spots to be empty, you replace one of the lines with a space.

Up to this point in the code, it's just letters. Somehow those letters have to be changed into `Items` and `Blocks`. That is done just below the crafting grid shape. After the last row that ends with a comma, the letter X is surrounded by apostrophes, which turns it into a character. After this character you need another comma, followed by the `Block` that you want it to represent, which are leaves in this example. The recipe created by the preceding code would require each area in the crafting grid to be filled with a leaf and it will then return an apple.

Alternative Shapes

Sometimes you don't require a three-by-three space to create a crafting recipe. In some cases you just need a two-by-two area. This is done using the following code.

```
"AB",
"CD",
```

With this code, the two-by-two shape can be placed anywhere in the crafting table. Another example of a differently shaped recipe is bread, which would look something like this:

```
"AAA",
```

In here the letter is the same each time, because bread is crafted using three wheat, but you can use different letters.

If you have more than a single `Item` or `Block` that is used in the shape, you have to add a comma behind the last `Item` or `Block`, write another pair of apostrophes, and follow that up by the other `Item` or `Block`.

An example of using a smaller grid, empty spaces, and using multiple components is shown next:

```
GameRegistry.addRecipe(new ItemStack(Items.arrow),
        "YZ",
        "X ",
        'X', Items.flint, 'Y', Items.stick, 'Z', Blocks.leaves
);
```

This will allow you to craft an arrow in your player inventory using leaves. Figure 3.2 shows the crafting of the recipe.

FIGURE 3.2
A smaller recipe in the player's inventory.

Another thing you may want to have in your recipes is that the returned `ItemStack` has a size of more than 1. Two other things that may be interesting for a recipe are metadata for the used and the resulting `Item` or `Block`. You can see all of these things in a single recipe, as follows:

```
GameRegistry.addRecipe(new ItemStack(Items.dye, 2, 1),
        "XY",
        'X', Items.redstone, 'Y', new ItemStack(Items.dye, 1, 1)
);
```

Each of the `ItemStacks` used in this recipe have three parameters now. The first is still an `Item` or `Block`. The second is the stack size. You can return a stack size of more than one, but you can't

use a higher stack size in a used item. The third parameter is the damage or metadata of the item. Metadata starts counting at 0 and goes up to 15. This recipe will combine one redstone and one red dye to make two red dyes. Red dye is the second metadata for dyes. Therefore the damage is 1.

This recipe may seem a little illogical, because this recipe will work only when the dye is on the right. For recipes like this, generally a Shapeless Recipe is used.

Creating a Shapeless Recipe

Shapeless recipes work similar to standard recipes. They are also set in `GameRegistry` and take the same parameters. The following code is the dye recipe redone in a shapeless recipe.

```
GameRegistry.addShapelessRecipe(new ItemStack(Items.dye, 2, 1),
       Items.redstone, new ItemStack(Items.dye, 1, 1)
);
```

In a shapeless recipe there is no need for letters or a crafting grid. You place up to nine `Item`, `Block`, or `ItemStack` objects in the `Object` array separated by commas, and the recipe will work. Except for that, it works exactly the same.

In game, this recipe can be crafted in the Player's inventory and looks like Figure 3.3.

FIGURE 3.3
A shapeless recipe.

Creating a Smelting Recipe

The final type of Recipe in Minecraft is a smelting recipe. A smelting recipe has only a single input and a single output slot. The input can, again, have a stack size of only one, and the output

can be more than one. Another thing to note is that smelting a recipe grants experience. The amount has to be set for the smelting recipe as well. Following is an example of a smelting recipe:

```
GameRegistry.addSmelting(Blocks.stone, new ItemStack(Blocks.stonebrick), 0.1F);
```

The first parameter can be an `Item`, `Block`, or `ItemStack`, but always with size 1. The second parameter has to be an `ItemStack`, and you can do everything with it you want here. The final parameter is a float value for the experience points you get from smelting this. A float in Java is basically a number with decimals. The vanilla smelting recipes have an experience return between 0.1F and 1F, where 0.1F is the recipe for cobblestone into stone and 1F is for smelting diamond ores.

Figure 3.4 shows this recipe in the game.

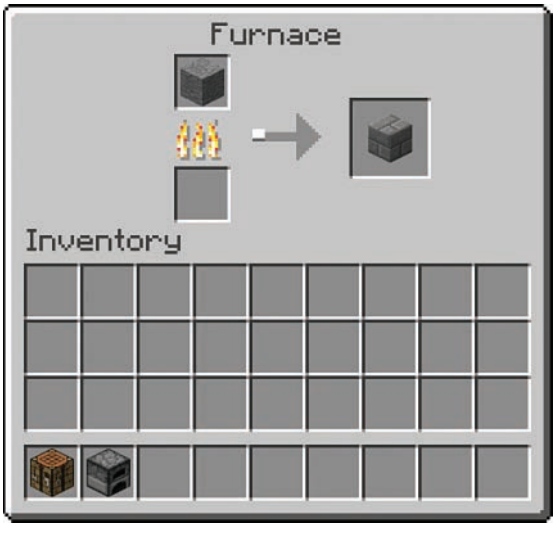

FIGURE 3.4
A smelting recipe.

Using Special `ItemStacks`

It is possible to use a special `ItemStack` in a recipe. To do that, you first have to create an `ItemStack` object and then use that in the recipe. Following is an example that creates a stone sword with sharpness 1 from a stone sword and a flint. This code uses a new file, so be sure to import it.

```
ItemStack enchantedSwordItemStack = new ItemStack(Items.stone_sword);
enchantedSwordItemStack.addEnchantment(Enchantment.sharpness, 1);

GameRegistry.addShapelessRecipe(enchantedSwordItemStack,
       Items.flint, Items.stone_sword
);
```

This code first creates an object of a normal stone sword. It then adds the sharpness enchantment of level 1 to it. Next, a shapeless recipe is added with the enchanted stone sword as the resulting item. The sword is crafted using a flint and a normal stone sword.

Figure 3.5 displays this recipe in game.

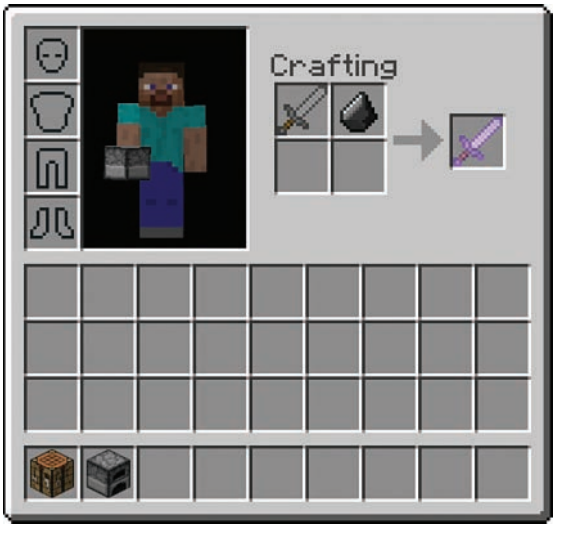

FIGURE 3.5
An enchantment recipe.

TRY IT YOURSELF

Creating a Knockback Sword

Try to create a sword with knockback using the code displayed previously, follow these three steps.

1. Create an `ItemStack` of any kind of sword.
2. Add the `knockback` enchantment to it.
3. Create a recipe using an unenchanted version of the sword along with gunpowder, with the returned item being the enchanted sword.

The code answer:

```
ItemStack knockbackItemStack = new ItemStack(Items.stone_sword);
knockbackItemStack.addEnchantment(Enchantment.knockback, 1);

GameRegistry.addShapelessRecipe(knockbackItemStack,
       Items.gunpowder, Items.stone_sword
);
```

Changing the Mob Spawn in a Dungeon

If you don't know what a dungeon is, you should read the following web page: http://minecraft.gamepedia.com/Dungeon.

Forge is a very good API. It lets you change many small things without changing any of the code directly, which results in mods not working together.

Removing a Mob

Forge allows you to change the mob spawn in a dungeon, for example, using a simple method in your mod file. First, a method that removes spider dungeons will look as follows.

Again, don't forget to import the new files.

```
DungeonHooks.removeDungeonMob("Spider");
```

This method removes the spider from the dungeon. It is a method in the `DungeonHooks` file that will select the mob that will be spawned in the dungeon. The `removeDungeonMob` takes a single `String` parameter that has to be the name of the mob in the code. If you want to know the exact names of all the mobs in Minecraft, go to the `EntityList` file in the net.minecraft.entity package of the ForgeSrc library.

Make sure you spell "Spider" correctly, or it won't work. The standard mobs spawning in a dungeon can be seen in the `DungeonHooks` file. You can also use this method to remove the spawn of a mob from a different mod, provided you have the name of it.

Adding a Mob

You can also add mobs to the dungeons using the next method:

```
DungeonHooks.addDungeonMob("Creeper", 100);
```

The first parameter is just like the `removeDungeonMob`, the code name of the mob. The second parameter is the spawn chance. As documented in the `DungeonHooks` file, spiders and skeletons have a 100 chance of spawning. Zombies have a chance of 200. Modifications such as this one can be found by browsing the Forge code.

TRY IT YOURSELF ▼

Adding a Mob

Add a different mob using the preceding method following these steps:

1. Find the mob name in the `EntityList` class, where all of the names are listed.
2. Add a method like the preceding one in your mod file with a 50 spawn chance.

Changing Chest Items

In Forge is another useful file that can be used to change the contents of basically any vanilla-generated chest.

Removing an `Item`

Following is an example of how to remove an `Item` from a chest:

```
ChestGenHooks.removeItem(ChestGenHooks.DUNGEON_CHEST, new ItemStack(Items.saddle));
```

`removeItem` is a method in `ChestGenHooks`, which takes two parameters. The first parameter is a `String`. This string specifies which chest it should remove an item from. You can manually type this, but it's suggested to use one of the variables in the `ChestGenHooks` file.

The second parameter is an `ItemStack` of the item that has to be removed from the chest.

Adding an `Item`

Just like with the dungeon mobs, you can also add new things to the chest here. The following code shows how to do that:

```
ChestGenHooks.addItem(ChestGenHooks.DUNGEON_CHEST, new
➥WeightedRandomChestContent(new ItemStack(Blocks.cobblestone), 25, 50, 10));
```

This method requires two parameters. Like with `removeItem`, the first parameter is the string that specifies the chest. The second parameter is a `WeightedRandomChestContent`. This is another file that requires four parameters.

The first parameter in the `WeightedRandomChestContent` is an `ItemStack` containing the `Item` or `Block` that should be generated. The second parameter is the minimum stack size in which it will generate. The third parameter is the maximum stack size. The final parameter is the chance of it being chosen. Gunpowder has a chance of 10 and a golden apple of 1.

The Results in the Mod File

After you have done all this, the mod file should look something like the following:

```
package com.wuppy.samsmod;

import net.minecraft.enchantment.Enchantment;
import net.minecraft.init.Blocks;
import net.minecraft.init.Items;
import net.minecraft.item.ItemStack;
import net.minecraft.util.WeightedRandomChestContent;
import net.minecraftforge.common.ChestGenHooks;
```

```
import net.minecraftforge.common.DungeonHooks;
import net.minecraftforge.fml.common.Mod;
import net.minecraftforge.fml.common.Mod.EventHandler;
import net.minecraftforge.fml.common.event.FMLInitializationEvent;
import net.minecraftforge.fml.common.registry.GameRegistry;

@Mod(modid = SamsMod.MODID, version = SamsMod.VERSION)
public class SamsMod
{
        public static final String MODID = "wuppy29_samsmod";
        public static final String VERSION = "1.0";

        @EventHandler
        public void init(FMLInitializationEvent event)
        {
                // crafting recipes
                GameRegistry.addRecipe(new ItemStack(Items.apple),
                                "XXX",
                                "XXX",
                                "XXX",
                                'X', Blocks.leaves);
                GameRegistry.addRecipe(new ItemStack(Items.arrow),
                                "YZ",
                                "X ",
                                'X', Items.flint, 'Y', Items.stick, 'Z', Blocks.leaves);
                GameRegistry.addRecipe(new ItemStack(Items.dye, 2, 1),
                                "XY",
                                'X', Items.redstone, 'Y', new ItemStack(Items.dye, 1, 1));

                // shapeless recipes
                GameRegistry.addShapelessRecipe(new ItemStack(Items.dye, 2, 1),
                ➥Items.redstone, new ItemStack(Items.dye, 1, 1));

                // special recipes
                ItemStack enchantedSwordItemStack = new ItemStack(Items.stone_sword);
                enchantedSwordItemStack.addEnchantment(Enchantment.sharpness, 1);

                GameRegistry.addShapelessRecipe(enchantedSwordItemStack, Items.flint,
                ➥Items.stone_sword);

                ItemStack knockbackItemStack = new ItemStack(Items.stone_sword);
                knockbackItemStack.addEnchantment(Enchantment.knockback, 1);

                GameRegistry.addShapelessRecipe(knockbackItemStack, Items.gunpowder,
                ➥Items.stone_sword );
```

```
            // smelting
            GameRegistry.addSmelting(Blocks.stone, new ItemStack(Blocks.stonebrick),
            ➥0.1F);

            // dungeon changes
            DungeonHooks.removeDungeonMob("Spider");
            DungeonHooks.addDungeonMob("Creeper", 100);
            ChestGenHooks.removeItem(ChestGenHooks.DUNGEON_CHEST, new
            ➥ItemStack(Items.saddle));
            ChestGenHooks.addItem(ChestGenHooks.DUNGEON_CHEST, new
            ➥WeightedRandomChestContent(new ItemStack(Blocks.cobblestone), 25,
            ➥50, 10));
        }
}
```

This code adds several standard recipes, a smelting recipe, and some shapeless recipes. It also changes up the dungeons a little bit.

Except for this, the code is grouped up a little, and there are a few comments that quickly tell the reader what it does. It's suggested to add this to your own file as well.

Summary

In this hour you learned everything there is to know about recipes. You know what kind of recipes are in Minecraft, which type is used for each recipe in vanilla, and you also started changing the game by adding some unique recipes to the game. After this you learned how to edit chests and dungeons without changing any of the vanilla files.

Q&A

Q. Is there a maximum number of recipes you can add?

A. There is no reachable limit on the number of recipes; you can add as many as you want although it is limited by computer power.

Q. Is it possible to create recipes that take an item with a stack size of more than one?

A. Unless you create a custom crafting table, this is impossible.

Q. Are there other files like this in Forge?

A. There are many files in Forge that change simple small things. Look around the code a bit if you are looking for something specific. These were just commonly used once.

Workshop

Finally you have created something that changes the game. Answer the following questions and do the exercises to make sure you understand it.

Quiz

1. What is the code name of an Ocelot?
2. Does the following smelting recipe give the player a lot of XP?

```
GameRegistry.addSmelting(Items.gold_nugget, new ItemStack(Blocks.obsidian), 3F);
```

3. If you want to create a flint and steel recipe that uses a flint and an iron ingot, which kind of recipe is the best choice?

Answers

1. The name is “Ozelot.”
2. Yes, three times as much as smelting diamond ore.
3. The best choice is a shapeless recipe.

Exercises

Try to create some custom recipes and change up the dungeon spawns a little:

1. Create a recipe for grass using a dirt block and a vine block.
2. Create a smelting recipe for flint that uses a gravel block.
3. Add Enderman to the possible mob spawns in a dungeon.
4. Add diamond blocks to the possible items in a dungeon.

This page intentionally left blank

Index

Symbols

A

B

c

D

E

F

G

H

I

J

K

L

M

N

O

P-Q

R

S

T

U

V

W

X-Y-Z