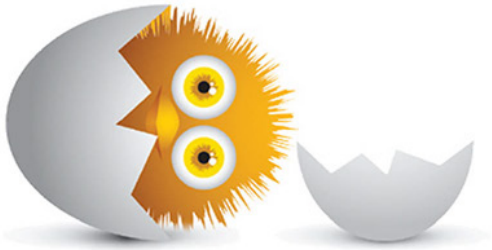
FULL COLOR

ABSOLUTE BEGINNER'S GUIDE

Minecraft® Mods Programming

No experience necessary!



Second Edition

Rogers Cadenhead

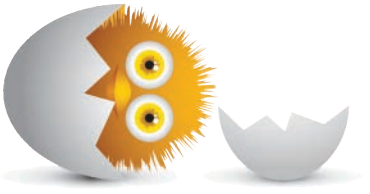
que

FREE SAMPLE CHAPTER

ABSOLUTE BEGINNER'S GUIDE TO

Minecraft® Mods Programming

No experience necessary!



Second Edition

Rogers Cadenhead

que®

800 East 96th Street,
Indianapolis, Indiana 46240

Absolute Beginner's Guide to Minecraft® Mods Programming

ISBN-13: 978-0-7897-5574-2
ISBN-10: 0-7897-5574-2

Library of Congress Control Number: 2015948680

Printed in the United States of America

First Printing: October 2015

Trademarks

Minecraft is a trademark of Mojang Synergies / Notch Development AB. This book is not affiliated with or sponsored by Mojang Synergies / Notch Development AB.

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Que Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Acquisitions Editor
Mark Taber

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Apostrophe Editing Services

Indexer
Lisa Stumpf

Proofreader
Sarah Kearns

Technical Editor
Boris Minkin

Publishing Coordinator
Vanessa Evans

Cover Designer
Matt Coleman

Compositor
Nonie Ratcliff

Contents at a Glance

This page intentionally left blank

Table of Contents

III Create Killer Minecraft Mods

About the Author

Rogers Cadenhead is a writer, computer programmer, and web developer who has written more than 20 books on Internet-related topics, including *Sams Teach Yourself Java in 24 Hours*. He maintains the Drudge Retort and other websites that receive more than 20 million visits a year. This book's official website is at www.javaminecraft.com.

Dedication

This book is dedicated to the kids out there who have been inspired by Minecraft to learn computer programming, whether they're 10, 20, or 50. There's a lot of great experiences ahead of you, not only in writing mods for a video game but in what you do with your skills beyond the game world.

Acknowledgments

To the folks at Pearson, especially Mark Taber, Andy Beaster, Lori Lyons, Boris Minkin, and San Dee Phillips. No author can produce a book like this on his own. Their excellent work will give me plenty to take credit for later.

To my wife, Mary, and my sons, Max, Eli, and Sam.

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: feedback@quepublishing.com

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

Reader Services

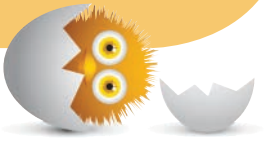
Visit our website and register this book at quepublishing.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

This page intentionally left blank

IN THIS CHAPTER

- Create a mod as a new project in NetBeans
- Add the Spigot API to the mod
- Write the Java code to implement the mod
- Configure a mod configuration file
- Organize a mod into the proper files and folders
- Build the mod in NetBeans
- Deploy the mod on a server
- Play the mod-ified Minecraft game



3

CREATE A MINECRAFT MOD

Now that you have a Spigot server for Minecraft set up and running and have installed the NetBeans integrated development environment (IDE), you're ready to create and deploy a mod.

Mods are special Java programs that run on a Minecraft server. They can't be run anywhere else.

Writing a mod requires the use of the Spigot API, a set of Java programs that do all of the background work necessary for the program to function inside a Minecraft game. A Java program also is called a *class*, so the Spigot API is called a *class library*.

The Spigot class library handles things like determining the (x,y,z) location of any object in the game, including a player. Everything you interact with in the game is represented in Spigot.

Before this book takes a full trip through Java, from the basics of the language into advanced features, this chapter demonstrates how a mod is created. This will give you a chance to see where all this material is headed. Many programming concepts will be unfamiliar to you, but all will be fully explained in subsequent chapters.

Creating Your First Mod

Mods are packaged as Java archive files, also called JAR files. NetBeans, the free integrated development environment from Oracle used throughout this book, automatically creates JAR files every time you build a project.

When you have finished writing a mod, you will be storing it under the server's folder in a subfolder named `plugins`.

The mod you are creating is a simple one that demonstrates the framework you'll use in every mod you create for Spigot. The mod adds a `/petwolf` command to the game that creates a wolf mob, adds it to the world, and makes you (the player) its owner.

CAUTION There are two very similar terms you encounter a lot when doing Minecraft programming: *mod* and *mob*. A *mod* is a Java program that runs on a server to add something cool to the game. A *mob* is any living creature in the game, such as a creeper, chicken, cow, cave spider, or catoblepas.

Actually, there are no catoblepases in Minecraft. I was just making sure you were paying attention.

Each mod will be its own project in NetBeans. To begin this project, follow these steps:

1. In NetBeans, select the menu command File, New Project. The New Project Wizard opens.
2. In the Categories pane, select Java, and in the Projects pane, select Java Application. Then click Next.
3. In the Project Name field, enter **`PetWolf`** (with no spaces and capitalized as shown).
4. Click the Browse button next to the Project Location field. The Select Project Location dialog appears.
5. Find the folder where you installed the Minecraft server. Select it and click Open. The folder appears in the Project Location field.

6. Deselect the Create Main Class check box.
7. Click Finish.

The PetWolf project is created, and two folders appear in the Projects pane, `Source Packages` and `Libraries`, as shown in Figure 3.1.

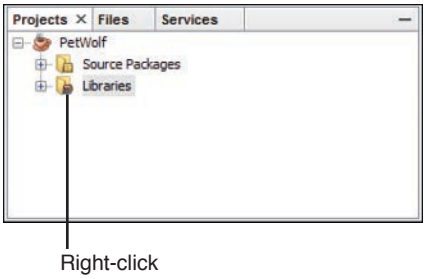


FIGURE 3.1

Viewing a project in the Projects pane.

On each mod project, you must add a Java class library before you begin writing Java code: the Spigot server's JAR file, which includes the Spigot API. Here's how to do this:

1. In the Projects pane, right-click the Libraries folder and select the menu command Add Library. The Add Library dialog opens.
2. Click the Create button. The Create New Library dialog appears.
3. In the Library Name field, enter **`Spigot`** and click OK. The Customize Library dialog opens.
4. Click the Add JAR/Folder button. The Browse JAR/Folder dialog opens.
5. Use the dialog to find and open the folder where you installed the server. You see a file in that folder named `spigotserver`.
6. Click that file.
7. Click Add JAR/Folder.
8. Click OK.
9. In the Add Library dialog, the Available Libraries pane now has a Spigot item. Select it and click Add Library.

The Projects pane now contains the JAR file for the Spigot API in the Libraries folder, so you're ready to begin writing the mod program.

Follow these steps to create the program:

1. Click File, New File. The New File wizard appears.
2. In the Categories pane, select Java.
3. In the File Types pane, select Empty Java File; then click Next.
4. In the Class Name field, enter **`PetWolf`**.
5. In the Package field, enter **`com.javaminecraft`**.
6. Click Finish.

The file `PetWolf.java` opens in the NetBeans source code editor.

Before you begin entering any code, this section explains the basics of how a mod is structured. Don't type in anything yet.

Every mod you create for Spigot begins with a framework of standard Java code. Here's the main part of that code, customized in a few places for this project:

```
public class PetWolf extends JavaPlugin {
    public static final Logger LOG = Logger.getLogger(
        "Minecraft");

    public boolean onCommand(CommandSender sender,
        Command command, String label, String[] arguments) {

        if (label.equalsIgnoreCase("petwolf")) {
            if (sender instanceof Player) {
                // do something cool here
                LOG.info("[PetWolf] Howl!");
                return true;
            }
        }
        return false;
    }
}
```

Looking at this framework, the only things that will change when you use it for a different mod are the three things that refer to PetWolf because those are specific to this project:

- The name of the program is `PetWolf`.
- The argument inside `label.equalsIgnoreCase("petwolf")` is the command the user will type in the game to run the mod. This program implements the command `/petwolf` (commands in a mod are preceded by a slash (/) character). The `label` object, which is sent as an argument to `onCommand()`, is a string that holds the text of a command entered by the user.
- The statement that calls `log.info("[PetWolf] Howl!")` sends a log message that is displayed in the Minecraft server window.

Everything a mod does when its command is entered goes at the spot marked by the comment `// do something cool here`. *Comments* are messages in a program that explain what it does to humans reading the code. They're ignored by the computer when the program runs.

The first thing the PetWolf mod needs to do is learn more about the game world, using these three statements:

```
Player me = (Player) sender;
Location spot = me.getLocation();
World world = me.getWorld();
```

A `Player` object called `me` is the character controlled by the person playing the game.

With this `Player` object, you can call its `getLocation()` method to learn the exact spot where the player is standing. A *method* is a section of a Java program that performs a task. Here, the method retrieves the player's current location as a `Location` object. Three things you can learn about a location are its (x,y,z) coordinates on the three-dimensional game map.

The `Player` object has a `getWorld()` method that responds with the `World` object that represents the entire game world.

Most of the mods you create need these three `Player`, `Location`, and `World` objects.

This mod creates a new mob that's a wolf, using the `spawn()` method of the `World` object:

```
Wolf wolfie = world.spawn(spot, Wolf.class);
```

There's a class for every type of mob in the game. The two arguments to the `spawn()` method are the location where the wolf should be placed and the class of the mob to create.

This statement creates a `Wolf` object named `wolfie` at the same spot as the player.

The color of the wolf's collar is set in this statement:

```
cat.setCollarColor(DyeColor.PINK);
```

After the wolf has been created, the player becomes its owner by calling the wolf's `setOwner()` method with the `Player` object `me` as the only argument:

```
wolf.setOwner(me);
```

Now you can begin typing. Put all this together by entering Listing 3.1 into the source code editor and clicking the Save All button in the NetBeans toolbar (or select File, Save).

LISTING 3.1 The Full Text of `PetWolf.java`

```
 1: package com.javaminecraft;
 2:
 3: import java.util.logging.*;
 4: import org.bukkit.*;
 5: import org.bukkit.command.*;
 6: import org.bukkit.entity.*;
 7: import org.bukkit.plugin.java.*;
 8:
 9: public class PetWolf extends JavaPlugin {
10:     public static final Logger LOG = Logger.getLogger(
11:         "Minecraft");
12:
13:     public boolean onCommand(CommandSender sender,
14:         Command command, String label, String[] arguments) {
15:
16:         if (label.equalsIgnoreCase("petwolf")) {
17:             if (sender instanceof Player) {
18:                 // get the player
19:                 Player me = (Player) sender;
20:                 // get the player's current location
21:                 Location spot = me.getLocation();
22:                 // get the game world
23:                 World world = me.getWorld();
```

```
24:
25:                 // spawn one wolf
26:                 Wolf wolf = world.spawn(spot, Wolf.class);
27:                 // set the color of its collar
28:                 wolf.setCollarColor(DyeColor.PINK);
29:                 // make the player its owner
30:                 wolf.setOwner(me);
31:                 LOG.info("[PetWolf] Howl!");
32:                 return true;
33:             }
34:         }
35:         return false;
36:     }
37: }
```

The `import` statements in Lines 3–7 of Listing 3.1 make five packages available in the program: one from the Java Class Library and four from Spigot. *Packages* are groups of Java classes that serve a related purpose. For instance, the `org.bukkit.entity` package referenced in Line 6 is a group of classes for the mobs in the game (which in Spigot are called *entities*).

NOTE These classes are used in the PetWolf program: `Logger` from `java.util.logging.Logger`, `Location` and `World` from `org.bukkit`, `Command` from `org.bukkit.command`, `Wolf` and `Player` from `org.bukkit.entity`, and `JavaPlugin` from `org.bukkit.plugin.java`.

You learn more about these packages as you use them to create more mods beginning in Chapter 18, "Spawn a Mob."

The `return` statements in Lines 32 and 35 are part of the standard mod framework. A method in Java can return a value when its task is completed. Your mods should return the value `true` inside the `onCommand()` method when the mod handles a user command and `false` when it doesn't.

You have created your first mod, but it can't be run yet by the Spigot server. It needs a file called `plugin.yml` that tells the server about the mod.

This file is a YAML file, which you also can create with NetBeans using these steps:

1. Select File, New File. The New File dialog opens.
2. In the Categories pane, scroll down and select Other.

3. In the File Types pane, select YAML File and click Next.
4. In the File Name field, enter **`plugin`**. (Don't put `.yml` on the end; this is done for you by NetBeans.)
5. In the Folder field, enter **`src`**.
6. Click Finish.

A file named `plugin.yml` opens in the source code editor with two lines in it:

```
## YAML Template.
---
```

Delete these lines. They aren't needed in this file. Enter the text of Listing 3.2 into the file, and be sure to use the same number of spaces in each line. Don't use tab characters instead of spaces.

LISTING 3.2 The Full Text of This Project's `plugin.yml`

```
 1: name: PetWolf
 2:
 3: author: Your Name Here
 4:
 5: main: com.javaminecraft.PetWolf
 6:
 7: commands:
 8:     petwolf:
 9:         description: Spawn a wolf as the player's pet.
10:
11: version: 1.0
```

Replace `Your Name Here` with your own name. Telling people you wrote a mod is the first step toward becoming a legendary Minecraft coder.

To double-check that you have entered the spaces correctly, there are four spaces in Line 8 before the text `petwolf` and eight spaces in Line 9 before `description`.

The `plugin.yml` file describes the mod's name, author, Java class file, version, command, and a short description of what the command does.

This file must be in the right place in the project. Look in the Projects pane, where it should be inside the `Source Packages` folder under a subheading called `<default package>`. This is shown in Figure 3.2.

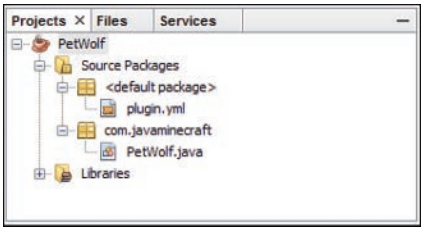


FIGURE 3.2

Checking the location of `plugin.yml`.

If the `plugin.yml` file is in the wrong place, such as under the `com.javaminecraft` heading, you can use drag and drop to move it to the proper location. Click and hold the file, drag it to the Source Packages folder icon, and drop it there.

You're now ready to build your mod. Select the menu command `Run, Clean and Build Project`. If this is successful, the message `Finished Building PetWolf (clean, jar)` will appear in the lower-left corner of the NetBeans user interface, way down at the bottom edge.

The mod is packaged as a file called `PetWolf.jar` in a subfolder of the project. To find it, click the Files tab in the Projects pane, expand the PetWolf folder (if necessary), and then expand the `dist` subfolder. The Files tab lists all the files that make up the project, as shown in Figure 3.3.

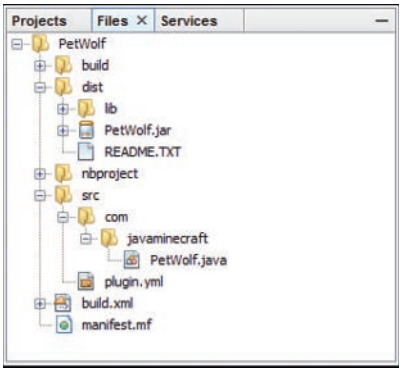


FIGURE 3.3

Finding the PetWolf mod's JAR file.

This `PetWolf.jar` file needs to be copied from the project folder to the Minecraft server. Follow these steps:

1. If the Minecraft server is running, stop it by going to the server window and typing the command **`stop`**; then press the spacebar or any other key to close that window.
2. Outside of NetBeans, open the folder where you installed the Minecraft server.
3. Open the `PetWolf` subfolder.
4. Open the `dist` subfolder.
5. Select the `PetWolf` file (a JAR file), and press Ctrl+C to copy it.
6. Go back to the Minecraft server folder.
7. Open the `plugins` subfolder.
8. Press Ctrl+V to copy `PetWolf` into it.

You have deployed your new mod on the Minecraft server. Start the server the same way you did before—by clicking the `start-server.bat` file you created. If you look carefully at the messages that display in the server window as the server loads, you see two new messages in the log file that display as it runs:

```
[PetWolf] Loading PetWolf v1.0
[PetWolf] Enabling PetWolf v1.0
```

These messages do not appear together. One appears close to the top and another close to the bottom.

If you don't see these messages, but instead see some long, complicated error messages, double-check everything in `PetWolf.java` against Listing 3.1 and `plugin.yml` against Listing 3.2 to ensure they were entered correctly; then rebuild and redeploy the mod.

TIP Still having problems making the PetWolf mod work? The source code, `plugin.yml` configuration file, and all other files for each book project can be found on www.javaminecraft.com. Visit the website to see files that were compiled successfully and run on a server.

After you run the Minecraft client and connect to your server, enter the command `/petwolf`. You now have a new wolf who will follow you around. Enter the command as many times as you like to keep adding wolves.

Figure 3.4 shows me and 20 wolves. Hostile mobs don't last long against us. We will rule this world. Hooooooooooooooooowl!

FIGURE 3.4

Your own wolf pack, courtesy of your own mod.

THE ABSOLUTE MINIMUM

This chapter was a sprint through the subject of how to write a Minecraft mod. The rest of the book will be at a much gentler pace that explains all the concepts in Java programming and the Spigot API that are necessary to create sophisticated mods.

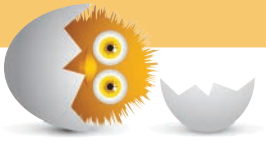
There are a lot of different ways to learn Java. Doing it inside Minecraft is one of the most entertaining. You will put your programming abilities to use in a three-dimensional world where you fight monsters, dig tunnels, build houses, and dodge the legions of the living dead—including some who ride chickens.

Minecraft is an excellent way to learn about object-oriented programming, one of the toughest aspects of the Java language to master. Everything you do in Java is accomplished with objects. You create them in constructors, give them knowledge

in variables, and tell them to do things by calling methods. Objects indicate the tasks they can perform by implementing interfaces.

All these concepts are new to you now, but as you learn them in the next 14 chapters, you will find they directly relate to mod programming. Need a zombie? Create a `Zombie` object. Want the player to move to a new spot? Call his `teleport()` method. Did you lose your horse? Call its `getLocation()` method.

So dig as deep into Minecraft mod programming as you like. You're going to use these skills far beyond the mines.



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