



Free Lesson Series

Beginning Minecraft Mods

Part Two: Minecraft Mods in Java

July, 2014

Copyright © 2014 by Homeschool Programming, Inc.

980 Birmingham Rd, Suite 501-128, Alpharetta, GA 30004

All rights reserved. No part of this document may be reproduced or transmitted in any form without written permission of the author.

LEGAL STUFF

(Nobody likes it, but you need to read it!)

Terms of Use

These lessons are copyright protected. Copyright © 2014 by Homeschool Programming, Inc. Using this document constitutes your agreement to the Terms of Use. You are not allowed to distribute any part of the lessons by any means to anyone else. You are not allowed to make it available for free (or fee) on any other source of distribution media, including the Internet, by means of posting or sending the file. If you wish to share, please link others to our website at <http://www.HomeschoolProgramming.com/minecraft>. You may reproduce (print or copy) course materials as needed for your personal use only.

Disclaimer

Homeschool Programming, Inc., and their officers and shareholders, assume no liability for damage to personal computers or loss of data residing on personal computers arising due to the use or misuse of this lesson material. Always follow instructions provided by the manufacturer of 3rd party programs that may be included or referenced by these lessons.

The material contained within these free lessons is not guaranteed. Minecraft is a rapidly evolving game and certain parts of the lessons may become outdated over time.

Copyright Notices

“Minecraft” and all related assets are copyright by Mojang AB.

Minecraft ®/TM & © 2009-2013 Mojang / Notch

“TeenCoder” and “KidCoder” are trademarks of Homeschool Programming, Inc.

Other Notices and Conditions

- These lessons are not officially associated with, or supported by Mojang in any way.
- These lessons are free (non-commercial). If you paid for them, you paid too much!
- You (the reader) are responsible for obtaining a legally licensed copy of your Minecraft game and abiding by all terms of the Minecraft EULA (see link below).

https://account.mojang.com/documents/minecraft_eula

Table of Contents

Legal Stuff	2
Terms of Use	2
Disclaimer	2
Copyright Notices	2
Other Notices and Conditions	2
Table of Contents	3
Welcome	4
About Us	4
Contact Us and Getting Help	4
Pre-Requisites – Skills You Need to Begin.....	5
Part Two: Minecraft Mods in Java	6
Lesson 5: Overview and Tools	7
Lesson 5 Activity: Installing Minecraft Forge	12
Lesson 6: Adding a Block.....	21
Lesson 6 Activity: Modifying the World Generator.....	33
Lesson 7: Adding a Recipe	41
Lesson 7 Activity: Your Own Recipes	47
Lesson 8: Adding a MOB.....	50
Lesson 8 Activity: Adding an Entity Texture	56
What’s Next?	65

WELCOME

Thanks for taking our free lessons on “Beginning Minecraft Mods – Part Two”! If you want to write your own Minecraft mods, these lessons will help you understand what is involved. Minecraft mods are a huge topic, and we don’t cover every angle in these lessons. But by the time you are done, you’ll understand the basics and have the resources you need to learn more. Please be sure you have completed “Part One” of these lessons before jumping into “Part Two”.

About Us



Homeschool Programming, Inc. provides computer science (programming) courses to 4th-12th grade students. Students in all learning environments (public school, private school, co-ops, and homeschool) with no prior experience can learn to write their own computer programs, websites, games, and apps. Please click on the links below for more information!

Course	Description
KidCoder: Web Design	Use HTML, CSS, and JavaScript to create web pages
KidCoder: Visual Basic	Learn Visual Basic to write your own programs and games
TeenCoder: C#	Use object-oriented concepts to write programs and games in C#
TeenCoder: Java/Android	Learn Java and Eclipse, study for the AP CS A exam, and write Android apps!

Contact Us and Getting Help

We welcome your questions and suggestions regarding these lessons or other programming topics! You may contact us through links provided on our website: <http://www.homeschoolprogramming.com>.

You can also follow us on Facebook and Twitter:



<http://www.facebook.com/HomeschoolProgramming>



<http://twitter.com/HSProgramming>

Pre-Requisites – Skills You Need to Begin

If you're an eager Minecraft player and have great ideas, that's a good start. But you need to have some extra skills up your sleeve before you begin writing your own mods.



Our lessons assume you already know how to do these things:

- You should be able to use a keyboard and mouse to select and run programs, use application menu systems, and generally work with your operating system.
- You should understand how to load and save files on the hard disk, and how to use the built-in operating system applications like Windows Explorer or Mac OS Finder to navigate a file system and directory structures and copy or move files and folders.
- You should have some practice using text editors such as Notepad or TextEdit.
- You should know how to use your web browser to find information on the Internet.
- You should understand ZIP files, how to create one, and how to expand one.
- You should have successfully completed “Part One” of our *Beginning Minecraft Mods* lessons.
- Of course, you should already know how to play Minecraft and have a licensed copy of the game.

A Special Note about Java...



Part One of this series can be done without any programming skills, but Part Two requires knowledge of the Java programming language and the Eclipse development environment. We assume you are **already comfortable** with Java and Eclipse; this is not a Java or Eclipse tutorial. If you need to learn these skills first, we invite you to check out our [TeenCoder: Java Programming](#) course. While the course is recommended for 9th-12th grade, younger students may also be able to complete the course with some extra help.

Age Ranges

These lessons are generally recommended for teens in the 9th-12th grade range. Younger students may also be capable of completing the lessons if they have all of the pre-requisite skills.

PART TWO: MINECRAFT MODS IN JAVA

Minecraft is a big, big game! The Internet is filled with websites, forums, and YouTube videos on many Minecraft topics. In fact, some of you probably know more about it than we do, so you are welcome to make suggestions to improve this or future lessons ☺. Our goal will be to go behind the scenes a bit and learn how to make some simple “mods” or modifications on your own. We are not going to talk about how to install or play Minecraft; you’ll have to figure that out on your own if you haven’t already.

In Part Two, you will learn how to change the fundamental behavior of the Minecraft game by creating new objects, recipes, and MOB’s. This part contains four separate lessons.

	Description
Lesson 5	Overview and Tools
Lesson 6	Learn how to add a new block to the environment
Lesson 7	Create new shapeless, shaped, and smelting recipes
Lesson 8	Learn how to add a new MOB to the game

Lessons 1 – 4, covering texture packs, are found in the separate “Part One” document. Please be sure you have successfully completed “Part One” before continuing to these “Part Two” lessons.

Lesson 5: Overview and Tools

It's going to be impossible to cover many parts of Minecraft development in a few short lessons. We are going to guide you through the installation of the software tools and some simple additions to the game. Serious Minecraft developers will want to make use of additional online resources for more guidance.

Links and Resources

Here are a few online links to get you started:

<http://minecraft.gamepedia.com/Mods>

http://minecraft.gamepedia.com/Mods/Creating_mods

<http://www.minecraftforum.net/forum/55-tutorials/>

Also, YouTube (www.YouTube.com) can be a great resource for a variety of tutorials and demonstrations, so if you get stuck or need help on a particular topic, you might find a video that provides an answer.

Understand that Minecraft versions are released frequently, and many resources you find online might refer to older versions. See our notes on Minecraft versions below.



We are going to give you many links to different Internet sites. To the best of our knowledge, these sites are basically safe. But ALWAYS use caution when visiting new areas, and have adult supervision if needed. Even safe-seeming sites may contain some salty language or other unwanted content, so visit at your own risk and responsibility! Don't click on anything you don't understand, and always scan downloaded programs with your anti-virus software before running them.

Minecraft Version History

New Minecraft releases happen all the time. In December 2013, version 1.7.4 was current. A few months later, version 1.7.10 is current. By the time you are reading this lesson, newer versions may be available.

You can find a summary of Minecraft versions here:

http://minecraft.gamepedia.com/Version_history

What does this mean for Minecraft mods? **Most mods will only work on one particular version.** So when you write a mod:

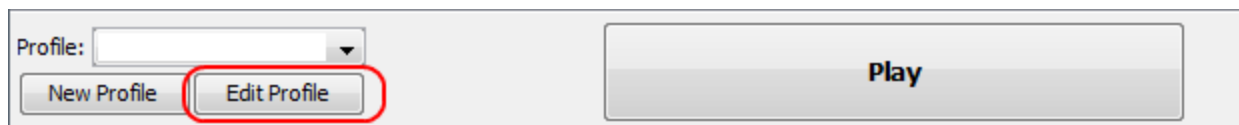
1. The software tools you use to create the mod are for a specific version
2. The mod you create will only run on that specific version
3. You need to run your own Minecraft client/server at that specific version

As of right now, the most recent “big” release that has attracted the most mod-work is 1.7.2. The software development tools that we’ll introduce shortly all work with version 1.7.2 and are not generally available for later versions yet. So we’ll be working with version 1.7.2 ourselves.

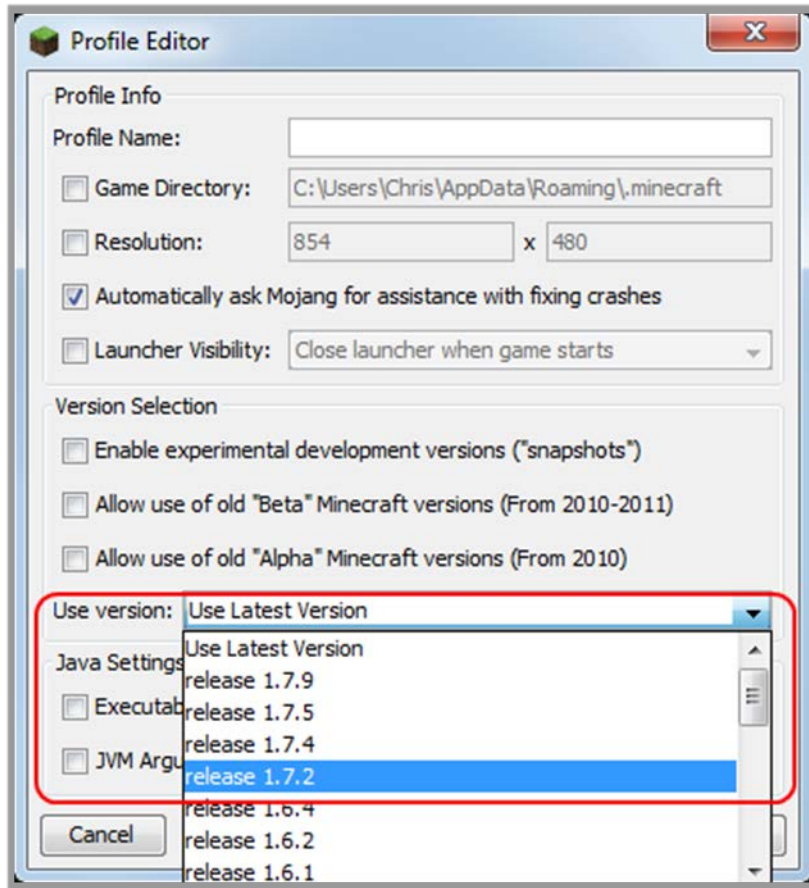
Running an Older Version

You may have purchased a more recent version of Minecraft, or have updated your own game to the latest versions. That’s OK! It’s actually possible to run your newer Minecraft game as an older version.

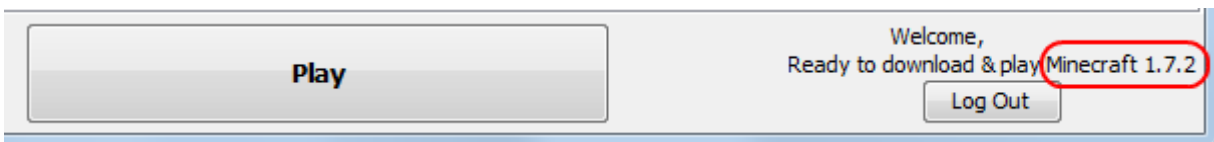
To select an older version, run Minecraft and click on the “Edit Profile” button next to the big “Play” button before you launch a game.



Then, in the “Profile Editor” pop-up, find the “Use version” combo box and select version 1.7.2:



Then click “Save Profile”. You should see the new version number to the right of the “Play” button on the launch screen.



Now when you click “Play”, you will be running that selected version 1.7.2.

Minecraft is a Client/Server Game

When you play Minecraft on a public server hosted by someone else, there are two main parts:

- **Client** – this is the Minecraft software running on your computer
- **Server** – this is the Minecraft software running on the host (server) computer

The client piece is what you see – the user interface. The server piece is relatively hidden and exists to keep track of the game data and ensure communication between all of the players.

When you run your own Minecraft game on your computer, you are actually running both the client and the server locally! You just don't see the server piece in action for the most part.

Mods for the Client

We are going to be focusing on mods for the client software. Most mods are client mods, and this is where you do things like create new blocks, recipes, and MOB's.

Mods for the Server

Server-side mods are mostly for administrators, and give useful admin features to make running the games easier.

If you have installed a mod on your client, and that mod is not also installed on the server, it will be ignored. So when you are playing a Survival Multi-Player game (SMP), your local client mods should not impact the server, unless that mod is also installed on the server.

Mod Tools

It may come as a surprise that mods are not officially supported by Minecraft! The game has been modified by individuals who have taken the official game, figured out how it works, and built a variety of tools to help you change the content. Now, the folks at Minecraft obviously know that mods are available, and mods are one of the reasons the game is so popular. But newer Minecraft versions will generally run ahead of the tools needed to make mods, and not all Minecraft versions are supported by the mod tools.

There are many types of mod tools out there, and some are more popular than others. For a brief list, see this link:

http://minecraft.gamepedia.com/Mods/Creating_mods

Minecraft Coder Pack

The Minecraft Coder Pack (MCP) is a set of tools that will let you decompile the Java bytecode from the official Minecraft game into readable Java source files. Those files can then be changed and re-compiled back into new Minecraft mods.

http://minecraft.gamepedia.com/Programs_and_editors/Minecraft_Coder_Pack

The MCP is often used as the basis for other, more advanced tools such as Minecraft Forge. When you use Forge, you will not have to download and install MCP separately; it will be bundled with the Forge tools.

Minecraft Forge

We are going to be using the **Minecraft Forge** tools, and their main website is here:

<http://www.minecraftforge.net/forum/index.php>

Forge provides a number of useful features, but most importantly Forge allows you to create mods without changing any of the built-in files that come with Minecraft. Forge also allows multiple mods to co-exist together on the same server instead of fighting one another.

The Forge installation process is described here:

<http://www.minecraftforge.net/wiki/Installation/Source>

There are also a series of videos on installation of the client tools here (Windows and Mac OS):

http://www.youtube.com/playlist?list=PLnpPguhVZMItSgWX8g8NFTq_rYrElxZ96

Forge can be used with the Eclipse Java development environment. You will be installing the Forge tools as your activity for this lesson, and then we'll start making some mods in the next lesson.

Lesson 5 Activity: Installing Minecraft Forge

In this activity we are going to get your computer set up with the Minecraft Forge tools for your Eclipse IDE. Before you begin:

- You must have installed the JDK
- You must have installed Eclipse

If you don't happen to have the JDK or Eclipse installed on your computer, a convenient set of instructions for installation of both pieces can be found here:

http://www.homeschoolprogramming.com/support/installing_teencoder_java_series.php

Understanding how to **actually use Eclipse to write Java code** is a pre-requisite to continuing with these lessons. Just following the installation instructions above will get the tools on your computer, but you need to have already learned how to write Java programs in Eclipse.

The main steps you will take for this activity are:

1. Download the “source” distribution of Minecraft Forge
2. Run some scripts from the command line that will download additional components
3. Set up your Eclipse workspace and run a test

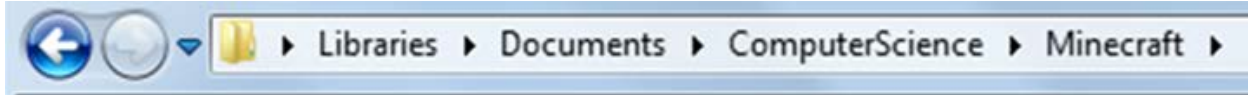
NOTE: If you encounter an error during any step in the process, your best bet is to carefully record the error message and do an online search for those keywords with “minecraft forge”, or read through the Forge forums. You are welcome to send those descriptions to us, but we'll simply take those exact research steps ourselves. There are many problems and solutions that have been encountered and resolved by others, and we don't necessarily have ready answers for every issue. Figuring things out is part of the “fun” of the Minecraft development process ☺.

Before You Begin

Before you start, make sure you launch at least one 1.7.2 game normally! Edit your Minecraft profile to select version 1.7.2, and actually launch a 1.7.2 game. This will ensure that all 1.7.2 components are installed on your system.

Create Your Working Directory

You are going to need a directory somewhere to hold all of your Minecraft development work and tools. This is NOT your “.minecraft” folder, but someplace else that you can remember. In our examples we’ll create a “ComputerScience” folder under our “My Documents” directory on Windows, or under our home folder on Mac OS. Then under “ComputerScience” we’ll create a “Minecraft” folder to hold the Forge tools and our new code.



OK, great, all of our Forge tools and new mod code will now live under this “Minecraft” folder.

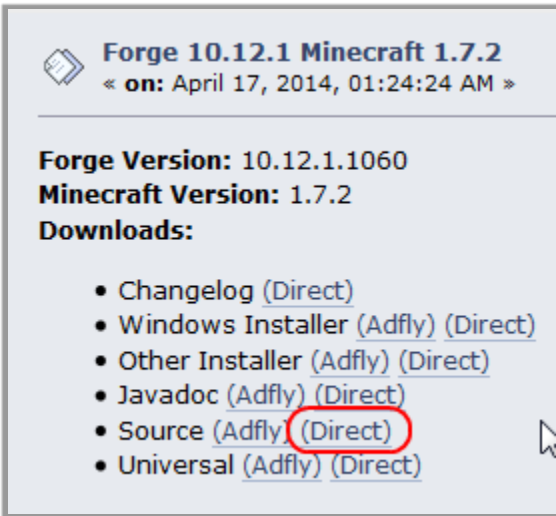
Download the “source” distribution of Minecraft Forge

Click on the link below to reach the Minecraft Forge Releases forum:

<http://www.minecraftforge.net/forum/index.php/board,3.0.html>

Select the highest “Forge 10.12.X Minecraft 1.7.2” link. There may be a later version than 10.12.1 by the time you do this – always pick the latest version **for 1.7.2!** Also ignore any later versions for higher versions of Minecraft such as 1.7.10.

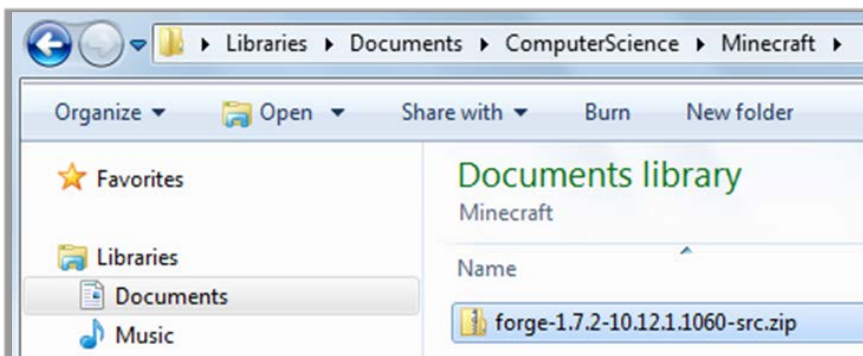
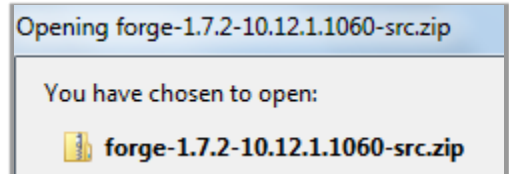




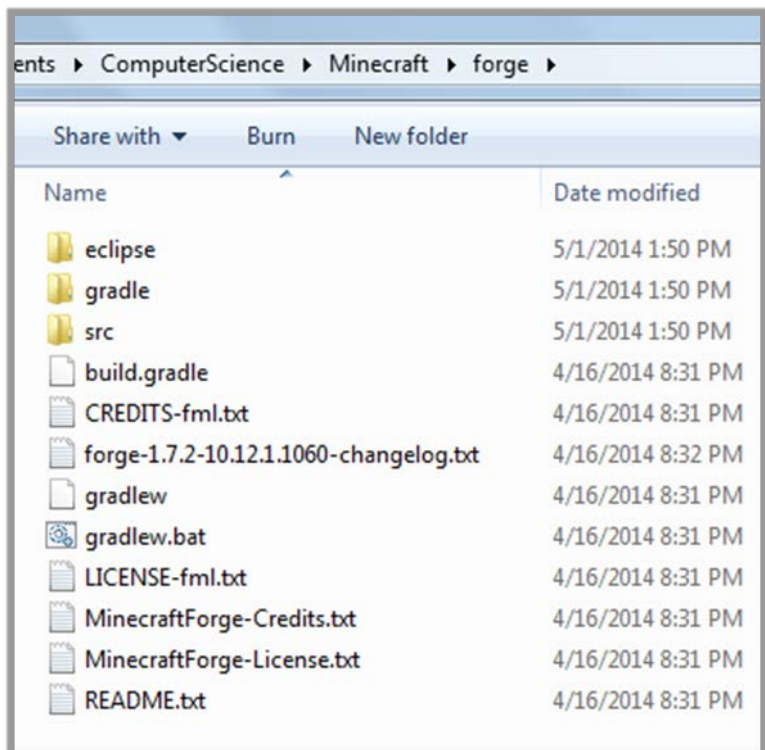
Next, click on the “Direct” link next to “Source”. As a developer, you need the source to can build your own mods.

Note: if you ever get routed to “Ad.fly” or asked to install some download manager program, you have clicked the wrong thing and strayed into “advertising” space. Back out and try again!

Save the resulting ZIP file to your working directory (such as “Computer Science/Minecraft”).



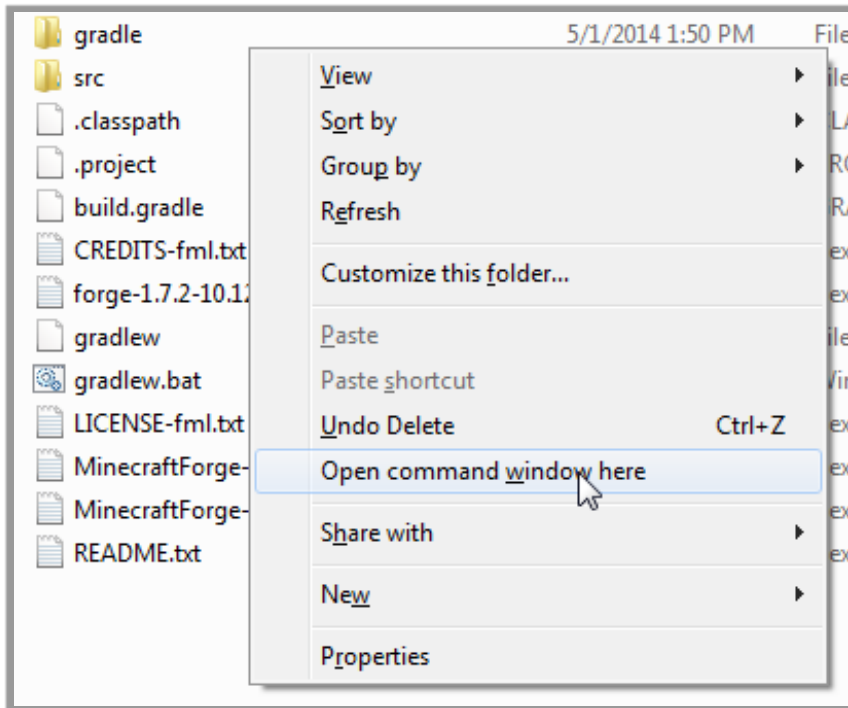
Then, un-zip the file to the “Forge” folder in your Minecraft directory.



Run Command-Line Configuration Scripts

Once you have the Forge source installed, you need to run some scripts from the command line in order to download, build, and configure all supporting component.

To open a command line in Windows, start in Windows Explorer and locate your new “Forge” directory. Hold down “SHIFT” and right-click with your mouse, then select “Open command line here”.



On Mac OS, you can just start your Terminal program, and then “cd” to the correct directly.

NOTE: You must run these scripts from within the “forge” directory. If your command line is showing some other current directory, you are in the wrong spot and the scripts will not run!

If you are new to command prompts, take a look at this document for advice on navigation and basic commands:

http://www.homeschoolprogramming.com/support/docs/instructions/Command_Prompts.pdf

The name of the script you will run is “gradlew”, and it has a number of optional parameters. Ideally you could run it with one or two steps, so start by typing in one of these commands:

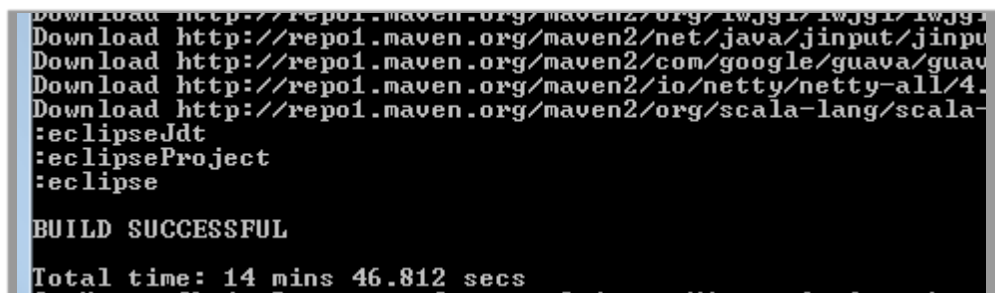
(Windows) `gradlew.bat setupDecompWorkspace --refresh-dependencies`

(Mac OS) `./gradlew setupDecompWorkspace --refresh-dependencies`



```
C:\windows\system32\cmd.exe
C:\Users\Chris\Documents\ComputerScience\Minecraft\forge>gradlew.bat setupDecompWorkspace --refresh-dependencies
```

Eventually you should get a SUCCESSFUL message after 10-15 minutes.



```
Download http://repo1.maven.org/maven2/org/iwjs1/iwjs1/iwjs1-
Download http://repo1.maven.org/maven2/net/java/jinput/jinput
Download http://repo1.maven.org/maven2/com/google/guava/guava
Download http://repo1.maven.org/maven2/io/netty/netty-all/4.
:eclipseJdt
:eclipseProject
:eclipse
BUILD SUCCESSFUL
Total time: 14 mins 46.812 secs
```

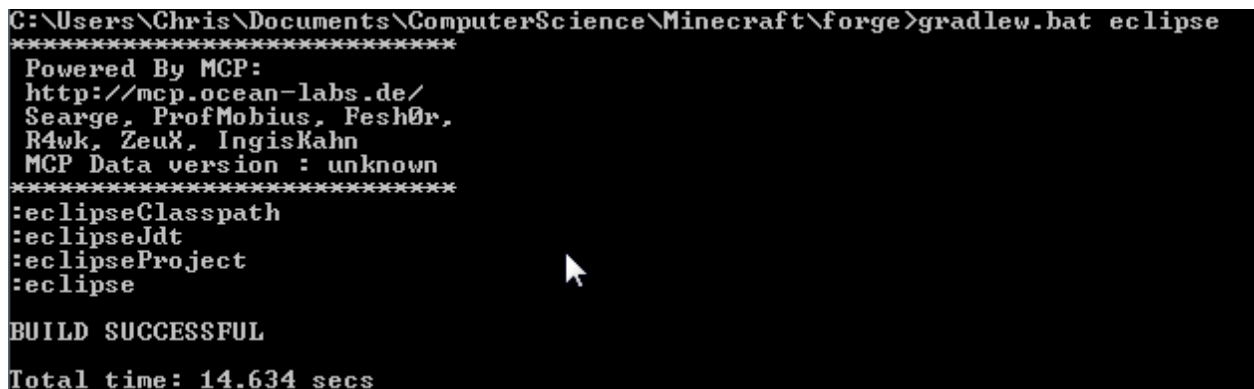
Then, assuming you get a SUCCESSFUL

message, type this next command to configure things for our Eclipse IDE:

(Windows) `gradlew.bat eclipse`

(Mac OS) `./gradlew eclipse`

Again this should take a while, and you will eventually get a SUCCESSFUL message.



```
C:\Users\Chris\Documents\ComputerScience\Minecraft\forge>gradlew.bat eclipse
*****
Powered By MCP:
http://mcp.ocean-labs.de/
Searge, ProfMobius, Fesh0r,
R4wk, ZeuX, IngisKahn
MCP Data version : unknown
*****
:eclipseClasspath
:eclipseJdt
:eclipseProject
:eclipse
BUILD SUCCESSFUL
Total time: 14.634 secs
```


If you see any build failures during this first step, try running this sequence instead:

(Windows)

```
gradlew.bat setupDecompWorkspace eclipse
```

```
gradlew.bat --refresh-dependencies
```

```
gradlew.bat setupDecompWorkspace
```

```
gradlew.bat eclipse
```

(Mac OS)

```
./gradlew setupDecompWorkspace eclipse
```

```
./gradlew --refresh-dependencies
```

```
./gradlew setupDecompWorkspace
```

```
./gradlew eclipse
```

There can admittedly be a bit of trial and error to get a successful build – the above sequence was used to correct our own build failures, and was found as a suggestion in a Forge forum!

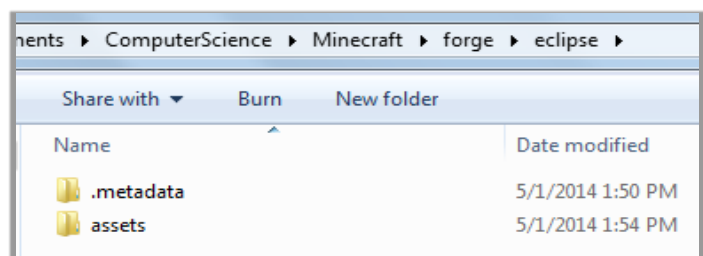
For more installation suggestions, you can check out this description and YouTube video:

<http://www.minecraftforge.net/wiki/Installation/Source>

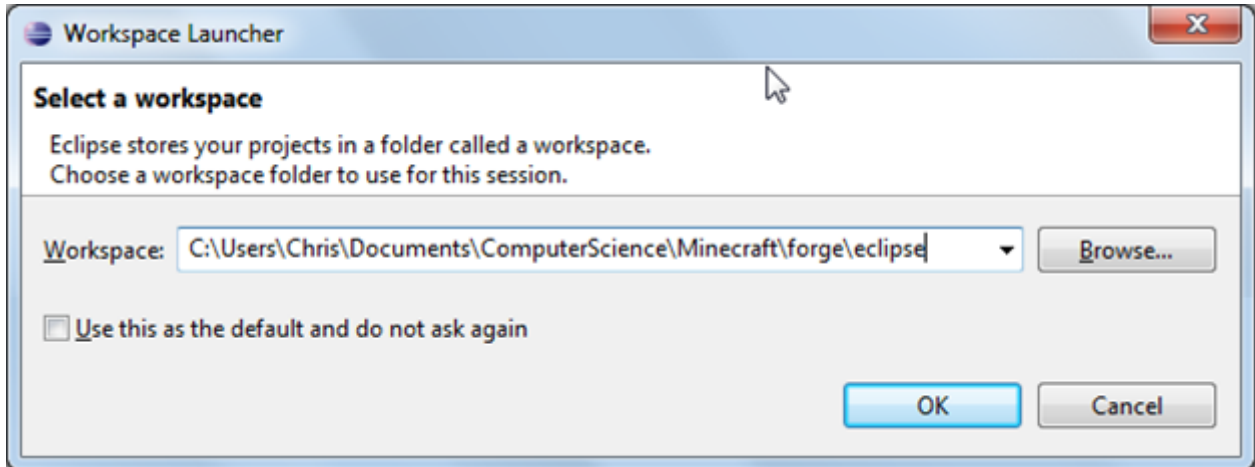
<https://www.youtube.com/watch?v=8VEdtQLuLO0&feature=youtu.be>

Set up your Eclipse workspace and run a test

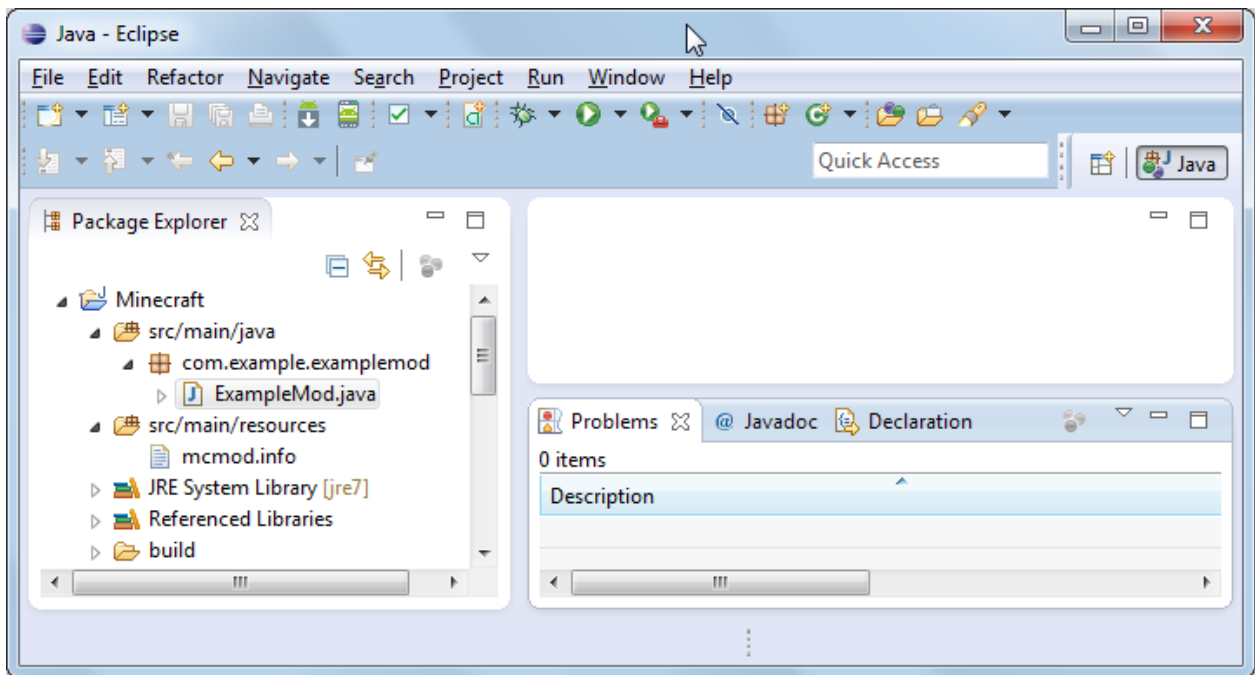
Next, you are going to run Eclipse and establish a NEW workspace! In your “forge” directory there should be an “eclipse” folder. This will be your new workspace, and it is already configured with a sample Minecraft mod project.



From Eclipse, select “File → Switch Workspace” and type in the path to your full “forge/eclipse” directory. Use the “Browse” button to navigate to the right directly if you don’t want to type everything in by hand. On Windows this would look similar to the example below:

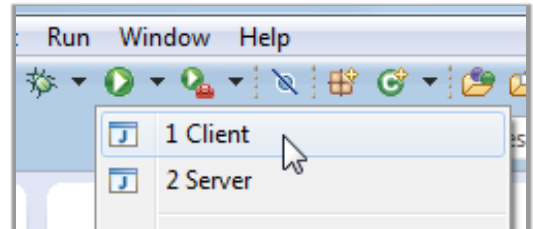


When Eclipse starts, you should see a “Minecraft” project already in place:



Give it a minute or two to ensure everything is automatically built. If you get any red errors anywhere, that is a sign that your command-line scripts were not successful, so you'll need to troubleshoot the issue with the help of some online research.

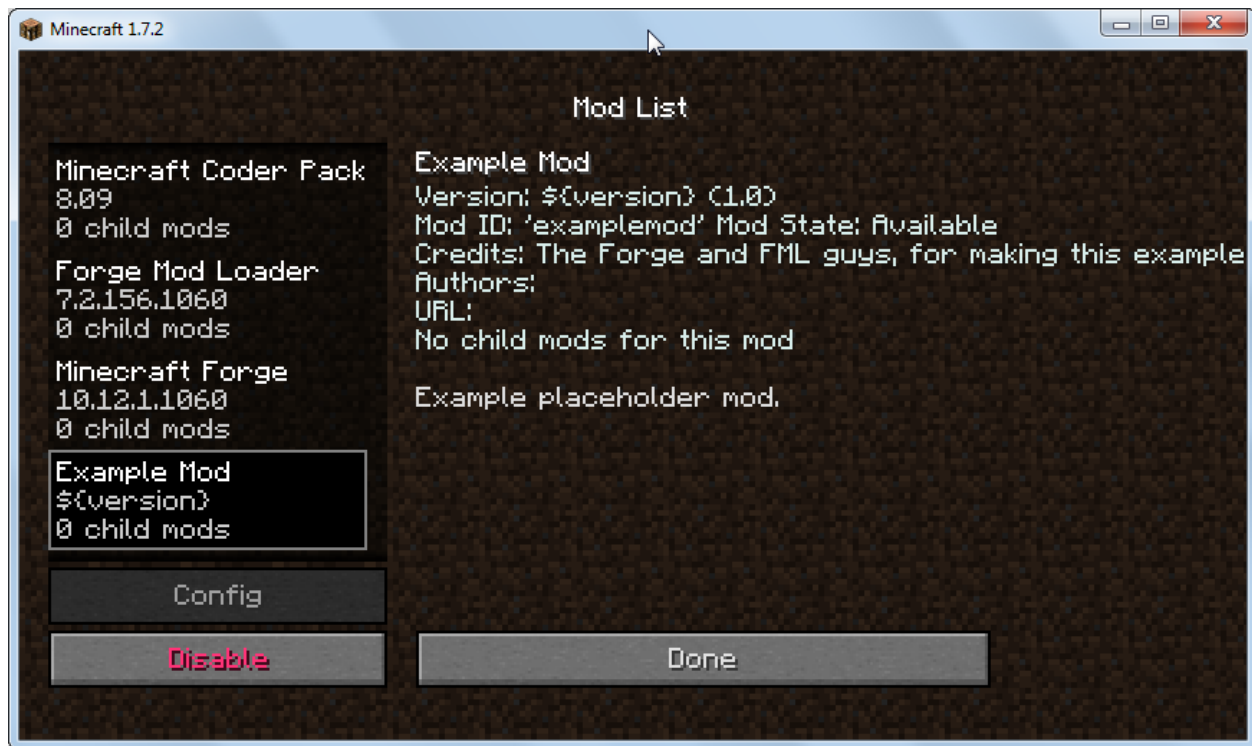
In order to launch Minecraft with Forge and the example mod already installed, click "Run → Client" from the Eclipse menu.



You should see Minecraft start, and you can see the Forge versions in the bottom left corner, plus a "Mods" button in the middle.



If you click on the “Mods” button you can see a list of things installed, which includes the Minecraft Coder Pack (MCP), Forge Mod Loader, Minecraft forge, and the Example Mod.



Congratulations! You are now all geared up and ready to begin developing your first mod. We'll begin exploring the things you can do in the next lesson.

Lesson 6: Adding a Block

In the last lesson you installed the Minecraft Forge mod tools, build a default mod, and saw it in action on your local Minecraft client. Now we are going to create a brand new type of block to place in our world!

The overall process for adding a new type of block includes:

1. Creating a new Java object representing your block
2. Creating new image files for your block
3. Creating a language file containing the name of your block
4. Registering your new block in your main Forge mod class

Tutorials, and Tutorials fixing Tutorials!

You can find mod steps described in many online tutorials. Unfortunately the Minecraft versions move so quickly that online tutorials quickly become out of date, even those hosted on the Forge site itself. Here is the starting point for the Minecraft Forge tutorials:

<http://www.minecraftforge.net/wiki/Tutorials>

These specific tutorials will get you started with basic modding concepts, creating new blocks, assigning an image (texture) to a block, and getting the block placed in your world.

http://www.minecraftforge.net/wiki/Basic_Modding

http://www.minecraftforge.net/wiki/Basic_Blocks

http://www.minecraftforge.net/wiki/Icons_and_Textures

http://www.minecraftforge.net/wiki/Tutorials/Ore_Generation

Some of the code in these tutorials does not compile correctly in the latest 1.7.2 Forge version, or steps may be left out or no longer needed. There is no way to know for sure until you try something and see if it works!

Fortunately, some individuals have begun to document what it takes to move from the earlier version 1.6.X mod code and tutorials to the newer 1.7.2 code. These two links lead to very useful clarifications of Forge documentation for 1.7.2.

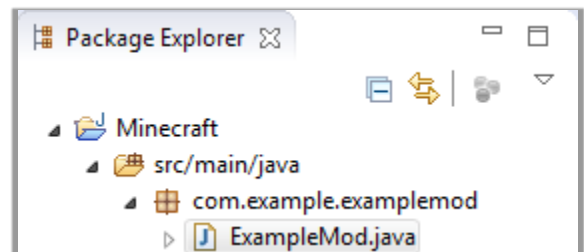
<http://www.wuppy29.com/minecraft/modding-tutorials/wuppys-minecraft-forge-modding-tutorials-for-1-7-updating-1-6-to-1-7-part-1-modfile-and-recipes/>

<http://www.minecraftforum.net/topic/2389683-172-forge-add-new-block-item-entity-ai-creative-tab-language-localization-block-textures-side-textures/>

We'll walk through some specific steps to create a new block in this lesson, so follow along in your Eclipse IDE to see the results on your computer. But you'll want to read a variety of online tutorials as well for more background and other ideas.

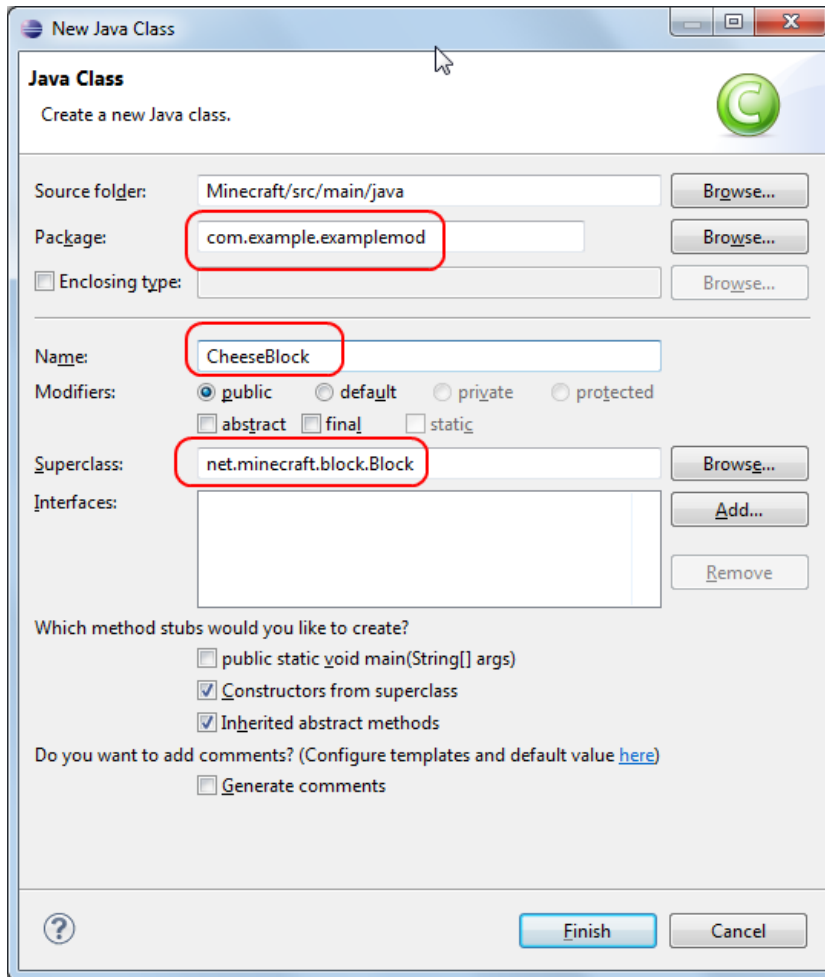
Creating a New Java Block Object

Let's begin by opening your Eclipse software and loading your Forge workspace from the last lesson. This workspace should contain a single "Minecraft" project that has an example mod named "ExampleMod".



Notice the example package name is "com.example.examplemod". If you are creating your own mod from scratch you can use a different package name, but we'll stick with this package for now. Any new classes you create for this lesson should belong to this same package.

We are going to create a new block called “**CheeseBlock**”. So the first step is to use Eclipse to create this Java class for us. From Eclipse, right-click on the “com.example.examplemod” package in the Package Explorer, and select “New → Class”.



Your **Package** should read “com.example.examplemod”. Type in the **Name** “CheeseBlock”, and also change the **Superclass** to read “net.minecraft.block.Block”.

You do **not** want to check the “public static void main(…)” checkbox.

Click “Finished” when done, and you should have a new “CheeseBlock.java” source file in your package.

Your initial “CheeseBlock.java” source file should look something like this:

```
package com.example.examplemod;

import net.minecraft.block.Block;
import net.minecraft.block.material.Material;

public class CheeseBlock extends Block {

    public CheeseBlock(Material p_i45394_1_) {
        super(p_i45394_1_);
        // TODO Auto-generated constructor stub
    }
}
```

Let’s make a couple of simplifications so this is more readable. Change the **Material** parameter name from “p_XXXXXX” to just read “**mat**”, and also make sure the matching curly braces are vertically aligned. You can remove the TODO comment also. Now your file looks like this:

```
package com.example.examplemod;

import net.minecraft.block.Block;
import net.minecraft.block.material.Material;

public class CheeseBlock extends Block
{
    public CheeseBlock(Material mat)
    {
        super(mat);
    }
}
```

Your “CheeseBlock.java” file should build without any errors after your changes.

Next, we want to configure the block with some properties. Inside the constructor, after the call to `super(mat)`, add the following lines:

```
public CheeseBlock(Material mat)
{
    super(mat);
    setHarvestLevel("shovel",0);
    setBlockName("cheeseBlock");
    setBlockTextureName("examplemod:cheeseBlock");
    setCreativeTab(CreativeTabs.tabBlock);
    setHardness(0.5F);
    setStepSound(Block.soundTypeGravel);
}
```

What are all these functions doing?

setHarvestLevel() – Allows you to control the type of tool needed to break (harvest) the block in survival mode. The first parameter is the name of the tool and the second is the level needed to make the block drop some ore.

setBlockName() – Gives your block a unique name that we’ll use to refer to the block elsewhere

setBlockTextureName() – Tells Forge that the image file for your block is named “cheeseBlock.png” and is stored in a particular directory (described later).

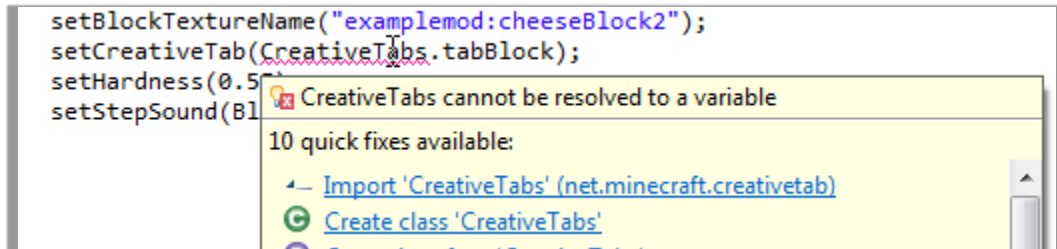
setCreativeTab() – This function tells Minecraft to show your block on a particular tab in Creative mode. We are choosing the “Block” tab in this example.

setHardness() – Sets the block hardness level, which should range from 0.0F to 50.0F. See http://www.minecraftwiki.net/wiki/Digging#Digging_speed for a description of hardness values.

setStepSound() – controls the sound you make when you walk on the block. There are many sounds available, so type “Block.” to see a pop-up list of sounds such as “Block.soundTypeGravel”.

You might see an error around the “CreativeTabs”...turn the page to fix it!

As you write new code, you may use objects from other packages that have not yet been imported into your source code. Eclipse will underline these objects in red and tell you the object cannot be resolved. If this happens, you can add the **import** statement at the top of your code manually. But Eclipse will do it for you also! Hover the mouse over the red error, and just click on the “Import...” line in blue underneath the “quick fixes” list. The **import** line will be added to your code, and the problem should be resolved.

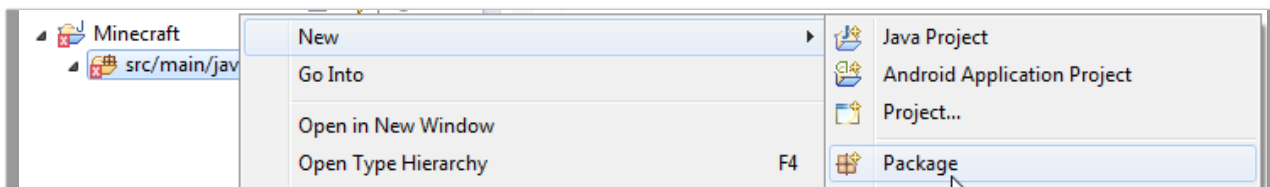


OK, your new **CheeseBlock** source code is complete; let’s turn our attention to the image.

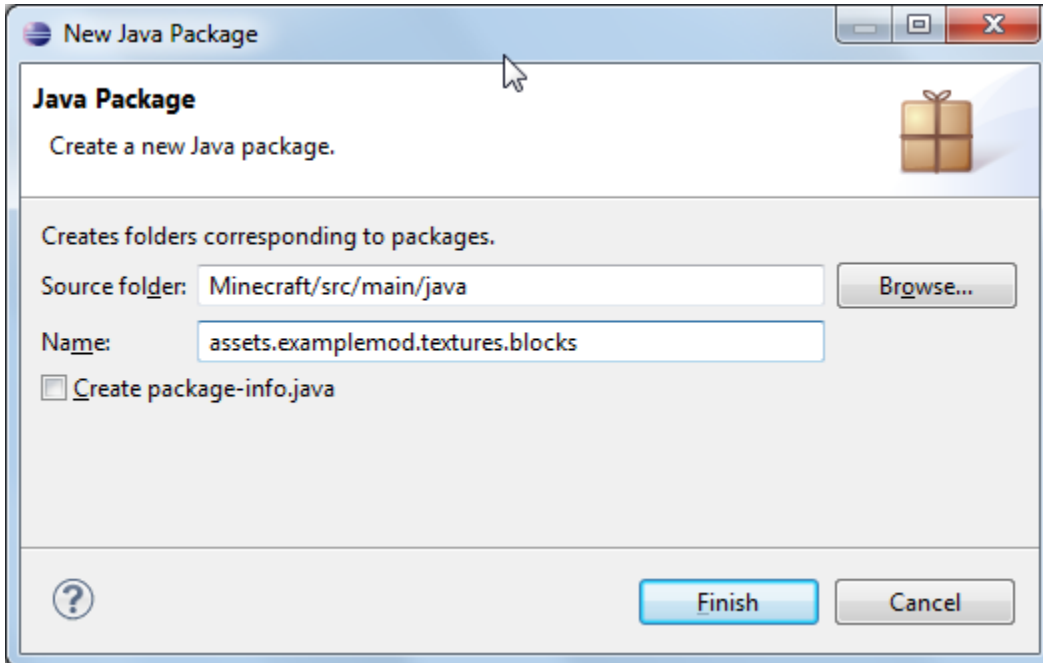
Creating a New Block Image

Most Minecraft images are 16 pixels wide and 16 pixels high. It is possible to assign different images to a block for a nicer 3D effect, but for starters we’ll just stick with a single image. When you are making a new custom block, it’s most fun to make your own image, so we’ll do exactly that.

Images must be placed in a specific directory structure matching the name of your mod. To create this directory, right-click on the “src/main/java” line underneath “Minecraft” in your Package Explorer and select “New → Package”.



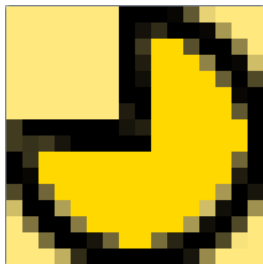
The package name should follow the pattern “assets.<modname>.textures.blocks”. <modname> in our case is “examplemod”. Use all lowercase letters, but otherwise match the name of your main mod class (ExampleMod).



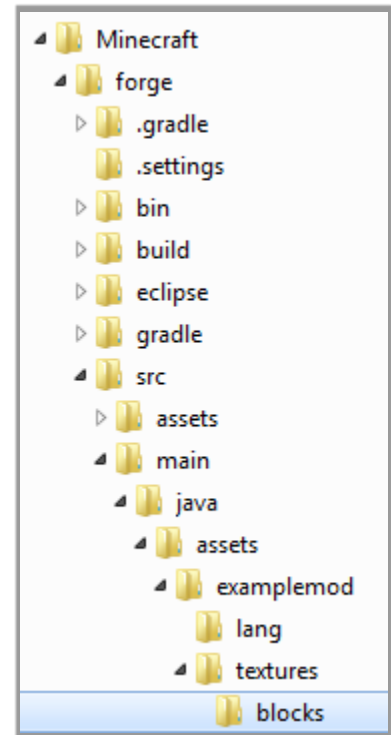
If you take a look at your file system in Windows Explorer or Mac Finder, you can see that an “assets/examplemod/textures/blocks” folder has been created under your “Minecraft/forge/src/main/java” directory.

This is where your new block image (texture) should go! We have created a 16x16 image file called “cheeseBlock.png” which you can download from here:

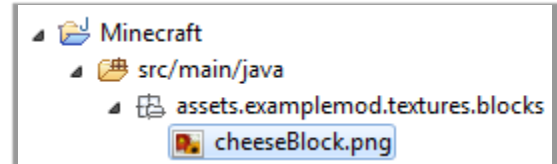
<http://www.homeschoolprogramming.com/downloads/cheeseBlock.png>



You may rightfully scoff at this artistic vision of cheese and substitute your own file. Just make sure it is 16 x 16 pixels and saved in PNG format in a file called “cheeseBlock.png”



Either way, save this file into the new “blocks” directory you just created. Then return to Eclipse and hit “F5” to force a refresh, and ensure the new file appears in your assets package.



Creating a Language File

Once your image file is in place, we need to make sure the new block will appear with a nicely formatted display name like “Cheese Block”. If you don’t follow these steps, Minecraft will show something ugly like “tile.cheeseBlock.name” instead.

Because Minecraft is available in different languages, all display strings are kept in separate files that can easily be swapped out for other languages. The language file is an asset and will be stored in a package structure similar to block images. Follow the same steps as when creating the new image assets (“New → Package”) to create a package called “assets.examplemod.lang”

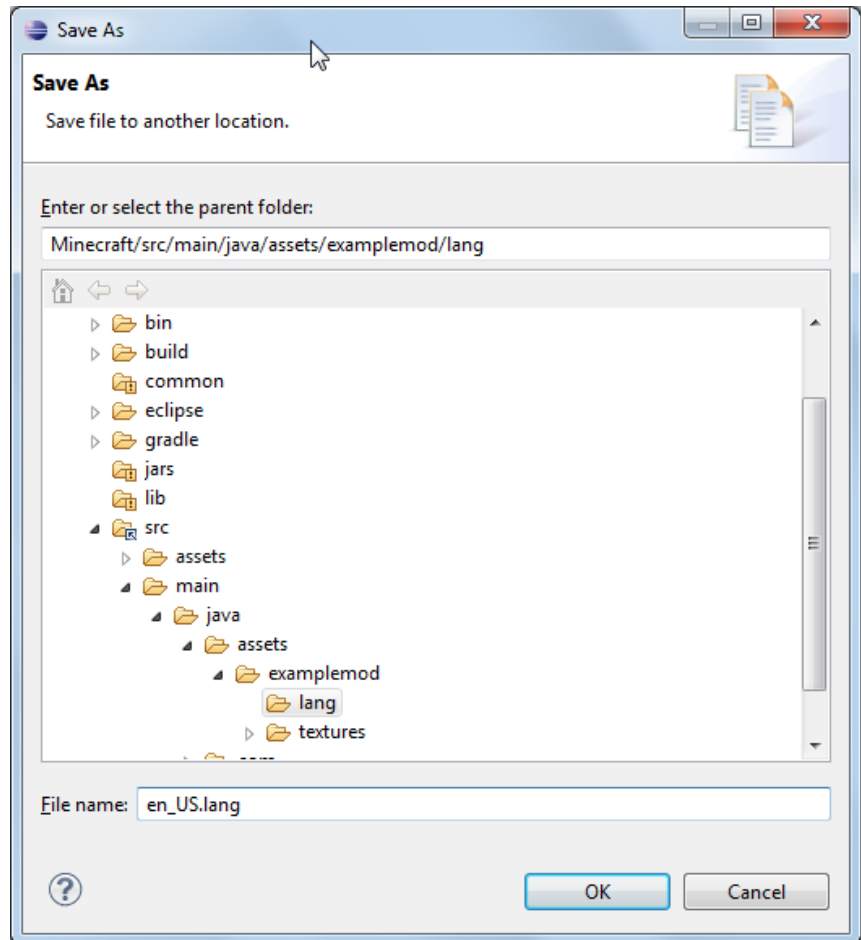
Once the directory has been created, you can right-click on the “assets.examplemod.lang” package and select “New → Untitled Text file”. This will give you an empty file called “Untitled”.

Enter the following line in the untitled file text editor:

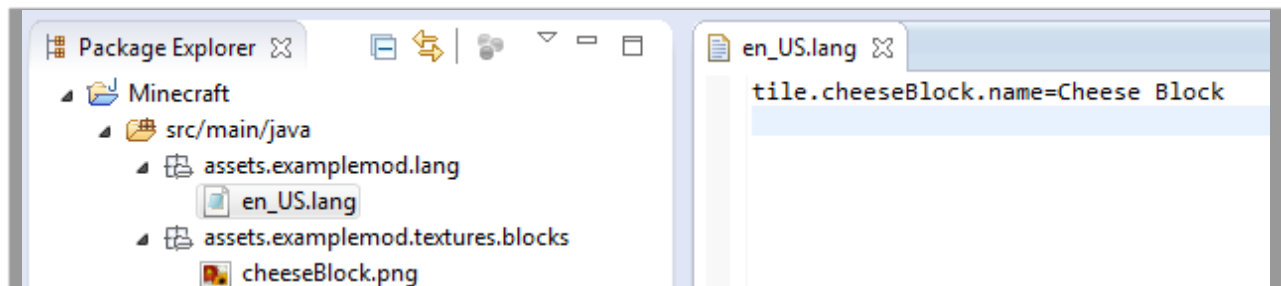
```
tile.cheeseBlock.name=Cheese Block
```

Now we need to save this file with the filename “en_US.lang” in our “assets.examplemod.lang” directory.

Click “File → Save As” and type in the “en_US.lang” filename, and be sure to select the new “assets.examplemod.lang” folder you just created in the previous step. Click “OK” when finished.



You should now see the “en_US.lang” file in your Package explorer with the contents shown below.



Registering Your New Block

There is one last step before we test everything! We need to let your main Mod class know that we have a new block that is available to the Minecraft environment. This is done by creating a new instance or copy of our custom block and then registering with a Forge object called **GameRegistry**.

Switch over to your “ExampleMod.java” file and add the lines shown below.

```
import net.minecraft.block.Block;
import net.minecraft.block.material.Material;
import cpw.mods.fml.common.registry.GameRegistry;
@Mod(modid = ExampleMod.MODID, version = ExampleMod.VERSION)
public class ExampleMod
{
    public static final String MODID = "examplemod";
    public static final String VERSION = "1.0";

    public static final Block cheeseBlock = new CheeseBlock(Material.ground);

    @EventHandler
    public void preInit(FMLInitializationEvent event)
    {
        GameRegistry.registerBlock(cheeseBlock, "cheeseBlock");
        System.out.println("CHEESE BLOCK >> " + cheeseBlock.getUnlocalizedName());
    }
}
```

The first change declares a new class variable called **cheeseBlock** and initializes it with a new **CheeseBlock**.

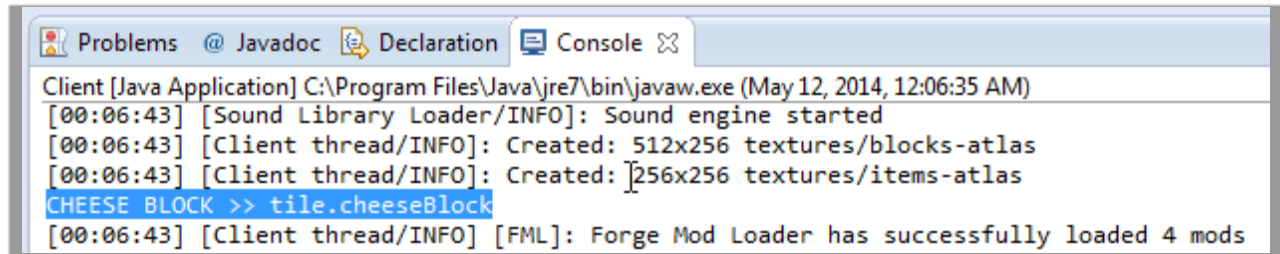
Next, if your initialization function is called “**init()**”, change it to read “**preInit()**” to make sure your changes are registered at the correct time.

Most importantly, call the **registerBlock()** function on the **GameRegistry** object and pass in the **cheeseBlock** variable you created, plus the block name “cheeseBlock”. You will need to import the **GameRegistry** package – let Eclipse do it for you!

Finally, just for fun, we printed out the cheese block name to the console. You can see this during startup of the client.

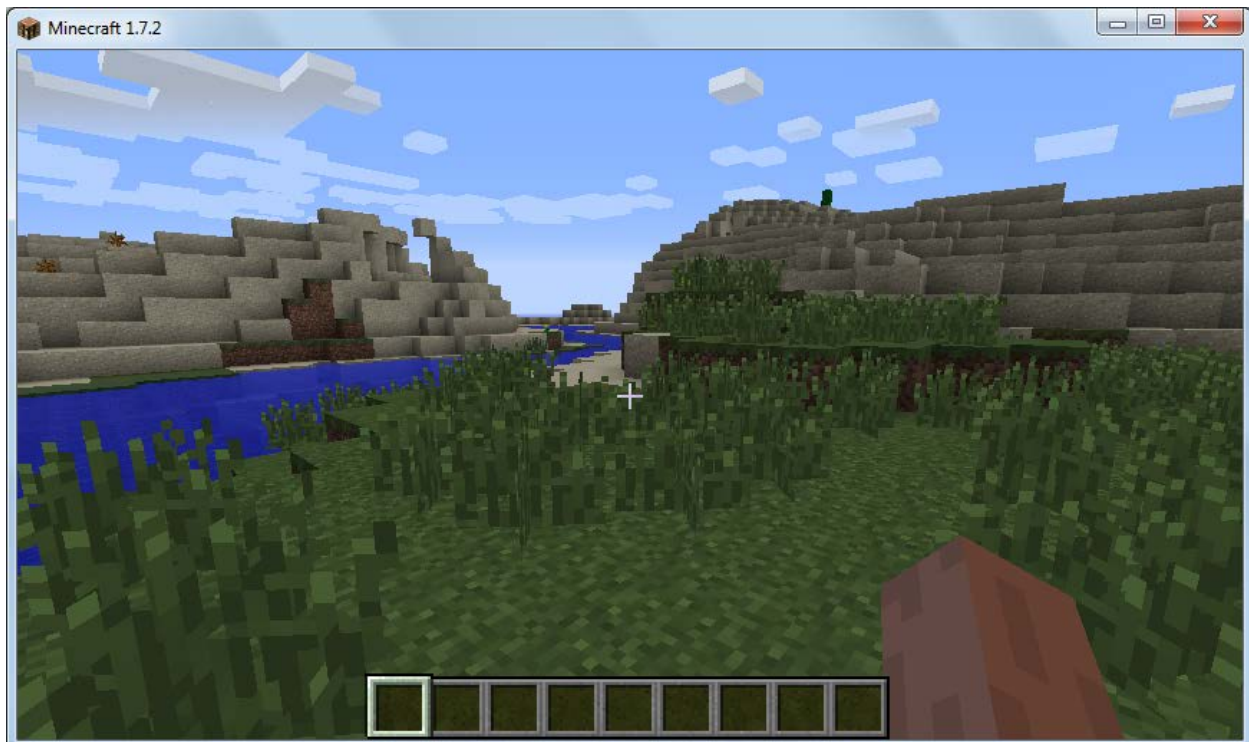
Testing Your New Block

It's time to test your new block! Everything should compile without errors at this point. So just click on the green arrow button in Eclipse to run the "Client" configuration. As the game starts you can see many status lines in the Eclipse Console, including anything you printed to the screen during initialization.



```
Client [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (May 12, 2014, 12:06:35 AM)
[00:06:43] [Sound Library Loader/INFO]: Sound engine started
[00:06:43] [Client thread/INFO]: Created: 512x256 textures/blocks-atlas
[00:06:43] [Client thread/INFO]: Created: 256x256 textures/items-atlas
CHEESE BLOCK >> tile.cheeseBlock
[00:06:43] [Client thread/INFO] [FML]: Forge Mod Loader has successfully loaded 4 mods
```

When Minecraft has started, choose "Single Player" and create a new world in "Creative" mode.



So far there is nothing to see in the environment, because Minecraft doesn't know how to use your new **CheeseBlock** when building the world. But you can hit the “E” button to bring up your inventory, click on the first “Building Blocks” tab, and scroll down toward the bottom to find our new “Cheese Block”!

In creative mode you have access to everything without having to go dig up or craft items, so this is a great way to test and ensure your block has been successfully coded and deployed.

For your lesson activity you are going to modify the terrain generation to include our new **CheeseBlock** in the world.



Troubleshooting

There are a number of parts and steps involved with even this simple mod, so it's easy to make a mistake. If things are not working as you expect, try these steps:

1. Carefully check your code for errors, including case-sensitivity issues. In general, assume everything is case sensitive and don't type “CheeseBlock” when you mean “cheeseBlock”.
2. Look at the console output in Eclipse when starting Minecraft. If you see an exception, try to understand the top-level error message – it may point you in the right direction!
3. Search online (forums, YouTube, etc.) for anyone that has encountered your specific problem and has offered a suggested fix.

Lesson 6 Activity: Modifying the World Generator

In this activity we are going to make the new CheeseBlock part of the environment so you can find and mine it in creative or survival mode. The two steps necessary for this are:

1. Create a new **WorldGenerator** object with the logic to distribute the block as an ore in the ground
2. Register this new **WorldGenerator** object with the main **ExampleMod** class so it will be used when the terrain is generated.

To begin, right-click on your “com.example.examplemod” package and create a new Java class (“New → Class”). Give the new class the name “**MyWorldGenerator**”, leave all other defaults alone, and click “Finish”. Your class won’t have much in it initially:

```
package com.example.examplemod;

public class MyWorldGenerator {

    public MyWorldGenerator() {
        // TODO Auto-generated constructor stub
    }

}
```

You can actually delete the constructor function, because we won’t be using it. Let’s also line up the curly braces vertically.

```
package com.example.examplemod;

public class MyWorldGenerator
{
}

}
```

OK, we want to add some code to this class. It must implement the **IWorldGenerator** interface and a **generate()** function with some specific parameters. You can cut-n-paste or type in the code directly as shown below.

```
public class MyWorldGenerator implements IWorldGenerator
{
    @Override
    public void generate(Random random, int chunkX, int chunkZ, World world,
        IChunkProvider chunkGenerator, IChunkProvider chunkProvider)
    {
        if (world.provider.dimensionId == 0)
        {
            for(int i = 0; i < 100; i++)
            {
                int firstBlockXCoord = chunkX*16 + random.nextInt(16);
                int firstBlockYCoord = random.nextInt(255);
                int firstBlockZCoord = chunkZ*16 + random.nextInt(16);

                WorldGenMinable minable = new WorldGenMinable(
                    ExampleMod.cheeseBlock, 50);
                minable.generate(world, random,
                    firstBlockXCoord, firstBlockYCoord, firstBlockZCoord);
            }
        }
    }
}
```

There are a number of objects that will need to be imported here, including **IWorldGenerator**, **Random**, **IChunkProvider**, **World**, and **WorldGenMinable**.

Use Eclipse to add **import** statements or type them in like this at the top of your file:

```
package com.example.examplemod;

import java.util.Random;

import net.minecraft.world.World;
import net.minecraft.world.chunk.IChunkProvider;
import net.minecraft.world.gen.feature.WorldGenMinable;
import cpw.mods.fml.common.IWorldGenerator;
```

OK, what’s going on with the main logic we added in the **generate()** function?

```
if (world.provider.dimensionId == 0)
{
    for(int i = 0; i < 100; i++)
    {
        int firstBlockXCoord = chunkX*16 + random.nextInt(16);
        int firstBlockYCoord = random.nextInt(255);
        int firstBlockZCoord = chunkZ*16 + random.nextInt(16);

        WorldGenMinable minable = new WorldGenMinable(
                                    ExampleMod.cheeseBlock, 50);
        minable.generate(world, random,
                        firstBlockXCoord, firstBlockYCoord, firstBlockZCoord);
    }
}
```

If the **world.provider.dimensionId** is zero, then we are talking about the normal surface world and not the Nether or End regions. For the surface world the **generate()** function will get called for each chunk of the terrain. We want to add in a random vein of “cheese” ore within that chunk. The three X, Y, and Z coordinates represent a random horizontal location within the chunk (X and Z) and a random height (from 0 to 255). A height of 64 is sea level. For more details on the Minecraft coordinate system, please see:

<http://minecraft.gamepedia.com/Coordinates>

So we are looping a number of times (e.g. 100) and calculating some random X, Y, and Z coordinates within that chunk as a starting point. We then create a new **WorldGenMinable** object using our **CheeseBlock** and 50 as the number of blocks in the vein. Finally, we call the `generate()` function on the **WorldGenMinable** object to create the random groupings of our **CheeseBlock** object.

You can play with the number of loops and the number of blocks in each vein. We have chosen some larger numbers to make it very obvious when the logic is working, so it will be easy to find this **CheeseBlock** underground. Just remember that the more you ask Minecraft to do when building the world, the longer it will take to generate the terrain.

As the last step, switch over to your “ExampleMod.java” file and find your `preInit()` function. Add in one more line to let Forge know to use your new world generator at startup:

```
GameRegistry.registerBlock(cheeseBlock, "cheeseBlock");  
GameRegistry.registerWorldGenerator(new MyWorldGenerator(),0);  
System.out.println("CHEESE BLOCK >> " + cheeseBlock.getUnlocalizedName());
```

OK you should be all set! Everything should build and run without errors. You can now create a new world (don't try an existing one) and let the new **WorldGenerator** do its thing. Continue to use Creative mode initially so we can easily move around the environment.

Here is a screen shot of one world after digging through some top layers. Buried yellow gold!



Sometimes veins of ore will be exposed on the sides of cliffs without any digging. So if you fly around a little bit, you can find some unburied cheese.



You should be able to also run the game in Survival mode and find and mine CheeseBlocks with your default tool. You'll see CheeseBlocks added to your inventory. We can't do anything with the cheese yet, but that's a subject for another lesson.

Activity Solution

This activity solution contains the fully coded source files if you get stuck on any step.

ExampleMod.java:

```
package com.example.examplemod;

import net.minecraft.block.Block;
import net.minecraft.block.material.Material;
import net.minecraft.creativetab.CreativeTabs;
import net.minecraft.init.Blocks;
import cpw.mods.fml.common.Mod;
import cpw.mods.fml.common.Mod.EventHandler;
import cpw.mods.fml.common.event.FMLInitializationEvent;
import cpw.mods.fml.common.registry.GameRegistry;

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

    public static final Block cheeseBlock = new CheeseBlock(Material.ground);

    @EventHandler
    public void preInit(FMLInitializationEvent event)
    {
        // some example code
        GameRegistry.registerBlock(cheeseBlock, "cheeseBlock");
        GameRegistry.registerWorldGenerator(new MyWorldGenerator(),0);
        System.out.println("CHEESE BLOCK >> " + cheeseBlock.getUnlocalizedName());
    }
}
```

CheeseBlock.java:

```
package com.example.exemplemod;

import net.minecraft.block.Block;
import net.minecraft.block.material.Material;
import net.minecraft.creativetab.CreativeTabs;

public class CheeseBlock extends Block
{
    public CheeseBlock(Material mat)
    {
        super(mat);
        setHarvestLevel("shovel",0);
        setBlockName("cheeseBlock");
        setBlockTextureName("exemplemod:cheeseBlock");
        setCreativeTab(CreativeTabs.tabBlock);
        setHardness(0.5F);
        setStepSound(Block.soundTypeGravel);
    }
}
```

en_US.lang:

```
tile.cheeseBlock.name=Cheese Block
```

MyWorldGenerator.java:

```
package com.example.examplerod;

import java.util.Random;

import net.minecraft.world.World;
import net.minecraft.world.chunk.IChunkProvider;
import net.minecraft.world.gen.feature.WorldGenMinable;
import cpw.mods.fml.common.IWorldGenerator;

public class MyWorldGenerator implements IWorldGenerator
{
    @Override
    public void generate(Random random, int chunkX, int chunkZ, World world,
        IChunkProvider chunkGenerator, IChunkProvider chunkProvider)
    {
        if (world.provider.dimensionId == 0)
        {
            for (int i = 0; i < 100; i++)
            {
                int firstBlockXCoord = chunkX*16 + random.nextInt(16);
                int firstBlockYCoord = random.nextInt(255);
                int firstBlockZCoord = chunkZ*16 + random.nextInt(16);

                WorldGenMinable minable = new WorldGenMinable(
                    ExampleMod.cheeseBlock, 50);
                minable.generate(world, random,
                    firstBlockXCoord, firstBlockYCoord, firstBlockZCoord);
            }
        }
    }
}
```


Lesson 7: Adding a Recipe

You just learned how to create a new block of cheese for our Minecraft mod. In this lesson we'll develop some new recipes! Your custom recipes can use any combination of existing blocks and new blocks you have added. There are three kinds of recipes you can create, and all three are pretty simple.

Getting Started

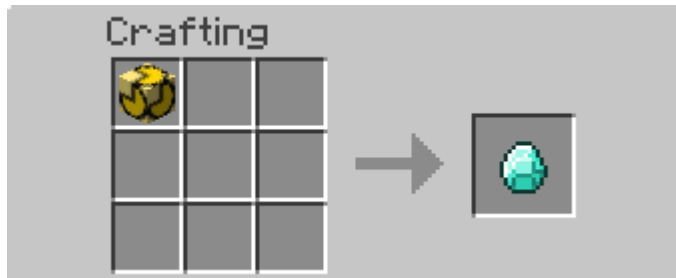
You should already be very familiar with built-in crafting and smelting concepts. In order to craft, you need a crafting table as shown by the block to the right. To smelt (transform one item into another), you need a furnace and some fuel as pictured to the left.



Crafting and smelting basically an exercise in putting the ingredients into your inventory, and then placing the correct arrangement of ingredients on the table to produce new items.

Shapeless Recipes

Shapeless recipes will create new items when the proper types and numbers of ingredients are in the crafting window, regardless of the positions. In the simple example below, we have created a shapeless recipe that takes our custom CheeseBlock as an input and produces a Diamond on output.



Pretty sweet, right?

You define new shapeless recipes in your main mod class (“ExampleMod.java”). The last three lines in the `preInit()` function below will do the trick!

```
public class ExampleMod
{
    public static final String MODID = "examplemod";
    public static final String VERSION = "1.0";

    public static final Block cheeseBlock =
        new CheeseBlock(Material.ground);

    @EventHandler
    public void preInit(FMLInitializationEvent event)
    {
        // some example code
        GameRegistry.registerBlock(cheeseBlock, "cheeseBlock");
        GameRegistry.registerWorldGenerator(new MyWorldGenerator(),0);

        // shapeless recipe to create diamonds from cheese
        ItemStack inputStack = new ItemStack(cheeseBlock);
        ItemStack outputStack = new ItemStack(Items.diamond);
        GameRegistry.addShapelessRecipe(outputStack, inputStack);
    }
}
```

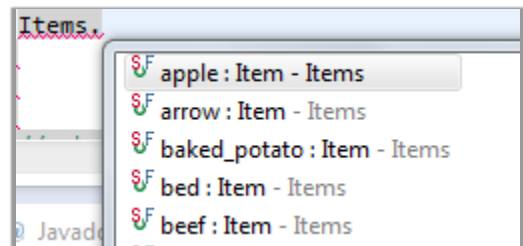
OK, what’s going on in these three lines of code?

```
ItemStack inputStack = new ItemStack(cheeseBlock);
ItemStack outputStack = new ItemStack(Items.diamond);
GameRegistry.addShapelessRecipe(outputStack, inputStack);
```

An **ItemStack** object can represent one or more copies of a particular block. You will need to import the **net.minecraft.item.ItemStack** and **net.minecraft.init.Items** objects at the top of your code. Don’t forget you can make Eclipse add these **import** statements for you when your mouse hovers over the object names.

```
import net.minecraft.item.ItemStack;
import net.minecraft.init.Items;
```

You create a new **ItemStack** object by passing an existing block object into the constructor. You can use a custom block that you already created in your mod, or you can pick any of the built-in blocks by typing “**Items.**” and then picking the target block from the pop-up list.



Finally, you call the **GameRegistry.addShapelessRecipe()** function and pass in two or more **ItemStack** objects. The first stack will be your output for the recipe, and the remaining stacks represent your input blocks. In our Cheese-to-Diamond example we only had one input block to produce the output. But you can add two or more stacks at the end to use more ingredients.

```
ItemStack cheeseStack = new ItemStack(cheeseBlock);
ItemStack potatoStack = new ItemStack(Items.potato);
ItemStack bakedStack = new ItemStack(Items.baked_potato);
GameRegistry.addShapelessRecipe(bakedStack,
                                cheeseStack, cheeseStack, potatoStack);
```

This example uses two cheese blocks and one potato block to make a baked potato.



Shaped Recipes

The next kind of recipe is a **shaped** recipe. With this kind you need to arrange the ingredients in a particular pattern on the crafting table to produce a result. You have a 3 x 3 grid to work with, so your shaped recipe can specify particular blocks (or empty spaces) in up to 3 rows and columns.



The function to create a shaped recipe is `GameRegistry.addRecipe()`. Again you'll use `ItemStack` to represent your output and input blocks. But you'll also need to add some parameters to specify the pattern on the grid. The parameters are a bit complicated, so let's look at an example.

```
ItemStack cheeseStack = new ItemStack(cheeseBlock);
ItemStack potatoStack = new ItemStack(Items.potato);
ItemStack bakedStack = new ItemStack(Items.baked_potato);
GameRegistry.addRecipe(bakedStack,
                        " x ", " x ", " y ",
                        'x', cheeseStack,
                        'y', potatoStack);
```

After the target stack (`bakedStack`), we have three strings: " x ",

" x ",

" y "

These strings represent the 3 x 3 grid on the table. The first string is the first row, and so on. A space in the string means that cell should be empty. A letter such as x, y, or z means that a specific block should be there. So in this example we have said one type of block (x) should be in the middle of the first two rows, and a second type of block (y) should be in the middle of the last row. After the rows are identified, then we pass in **pairs** of parameters that have a single-character letter (e.g. 'x') and the matching `ItemStack`. So 'x' is a cheese block and 'y' is a potato in this example.

As a result, our recipe demands two cheese blocks in the first two middle cells and a potato on the bottom middle cell. When that recipe is completed, we'll make a baked potato!



Make SURE you carefully provide all of the parameters you need, and that your pattern strings have exactly the right number of characters to represent the grid. Otherwise you may see an exception on startup.

Can you tell what pattern is represented by these strings?

"x "

" y "

" x"

We won't keep you in suspense. Here is the result, assuming the 'x' and 'y' blocks are still cheese and potatoes:



As you can imagine, you can create some pretty complicated recipes with up to 9 different items in patterns within the 3 x 3 grid.

Smelting

Smelting is the process of transforming one block to another on a furnace, powered by fuel. Since there is only one input ingredient, adding a new smelting recipe is very easy. You use the **GameRegistry.addSmelting()** function, passing in the input ingredient (Block or Item) and the output **ItemStack**. You also need to give an experience value between 0.0F and 1.0F.

Let's create a smelting recipe that will let you transform a brick into a diamond.

```
ItemStack diamondStack = new ItemStack(Items.diamond);  
GameRegistry.addSmelting(Items.brick, diamondStack, 0.5F);
```

As you can see, we simply pass in a reference to the input item or block, then our output **ItemStack**, and then an experience value. Now, when you place a brick in the furnace, with some fuel, it will produce a diamond!



You can basically transform any input to any other output using smelting.

Further Reading

Most of the recipes that make sense (such as smelting clay into a brick) have already been defined in the basic game. You can see this link for a list of recipes:

<http://minecraft.gamepedia.com/Crafting/CompleteList>

The examples we gave for crafting and smelting were mostly nonsensical (although changing cheese into a diamond seems like a great idea). In most cases you'll create your own custom outputs to go with your inputs, but to keep things short we used mostly built-in items.

For more information on crafting and smelting recipes, you can start with these links, and of course search YouTube and other forums.

http://www.minecraftforge.net/wiki/Crafting_and_Smelting

<http://www.wuppy29.com/minecraft/modding-tutorials/wuppys-minecraft-forge-modding-tutorials-for-1-7-updating-1-6-to-1-7-part-1-modfile-and-recipes/>

<http://www.minecraftforum.net/topic/2389683-172-forge-add-new-block-item-entity-ai-creative-tab-language-localization-block-textures-side-textures/>

Lesson 7 Activity: Your Own Recipes

In this activity, you are going to create three of your own recipes. You can use only built-in blocks and items or create your own custom input and output blocks. Use your imagination, the sky is the limit!

Shapeless Recipe

Create at least one shapeless recipe that takes two or more input **ItemStacks** to produce an output **ItemStack**.

Shaped Recipe

Create at least one shaped recipe that takes two or more input **ItemStacks** in a specific arrangement on a 3x3 grid and produces an output **ItemStack**.

Smelting Recipe

Create at least one smelting recipe that transforms one input **Block** or **Item** into an output **ItemStack**.

Testing Tips

You should build and test each of your recipes in turn, and make sure the first one works before going on to the next one. Remember that you just need to click the green “Run Client” arrow button in Eclipse to start Minecraft with your mod in place. You do NOT need to create a new world each time just to test out new recipes. So create one world in creative mode to give you easy access to all materials, and go ahead and place a crafting table and furnace nearby. Then you can quickly test changes by reloading that same world.

Activity Solution

Your activity solutions may vary widely, and can include custom Block or Item Java classes as part of the input and output ingredients. Information on creating custom Blocks can be found in the previous lesson. The file below shows the fully coded “ExampleMod.java” file with the **preInit()** method demonstrating each of the example recipes discussed in the lesson. These recipes can be used as templates for your mods.

```
package com.example.examplemod;

import net.minecraft.block.Block;
import net.minecraft.block.material.Material;
import net.minecraft.creativetab.CreativeTabs;
import net.minecraft.init.Blocks;
import net.minecraft.init.Items;
import net.minecraft.item.ItemStack;
import cpw.mods.fml.common.Mod;
import cpw.mods.fml.common.Mod.EventHandler;
import cpw.mods.fml.common.event.FMLInitializationEvent;
import cpw.mods.fml.common.registry.GameRegistry;

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

    public static final Block cheeseBlock = new CheeseBlock(Material.ground);

    @EventHandler
    public void preInit(FMLInitializationEvent event)
    {
        // some example code
        GameRegistry.registerBlock(cheeseBlock, "cheeseBlock");
        GameRegistry.registerWorldGenerator(new MyWorldGenerator(),0);

        // shapeless recipe to create diamonds from cheese
        ItemStack inputStack = new ItemStack(cheeseBlock);
        ItemStack outputStack = new ItemStack(Items.diamond);
        GameRegistry.addShapelessRecipe(outputStack, inputStack);
    }
}
```



```
// shapeless recipe to create a baked potato from 2 cheese and 1 potato
ItemStack cheeseStack = new ItemStack(cheeseBlock);
ItemStack potatoStack = new ItemStack(Items.potato);
ItemStack bakedStack = new ItemStack(Items.baked_potato);
GameRegistry.addShapelessRecipe(bakedStack,
                                cheeseStack, cheeseStack, potatoStack);

// shaped recipe to create a baked potato from 2 cheese and 1 potato
ItemStack cheeseStack2 = new ItemStack(cheeseBlock);
ItemStack potatoStack2 = new ItemStack(Items.potato);
ItemStack bakedStack2 = new ItemStack(Items.baked_potato);
GameRegistry.addRecipe(bakedStack2, "x ", " y ", " x",
                       'x', cheeseStack2, 'y', potatoStack2);

// shaped recipe to create diamonds from cheese and potatoes and coal
ItemStack coalStack = new ItemStack(Items.coal);
GameRegistry.addRecipe(new ItemStack(Items.diamond), "x ", " y ", "zzz",
                       'x', cheeseStack, 'y', potatoStack, 'z', coalStack);

// smelting recipe to create diamonds from bricks
ItemStack diamondStack = new ItemStack(Items.diamond);
GameRegistry.addSmelting(Items.brick, diamondStack, 0.5F);

System.out.println("CHEESE BLOCK >> " + cheeseBlock.getUnlocalizedName());
}
}
```

Lesson 8: Adding a MOB

This is our last Minecraft lesson in this series! You are going to learn how to create your own “**mob**”. The term “mob” is short for “**mobile**” or “**Mobile OBject**”, and it refers generally to any active person or critter that wanders around in a game. In Minecraft the technical term for mob is “**entity**”, so you may see both “mob” and “entity” used interchangeably.

Minecraft Entities

To create a new Minecraft mob with Forge in 1.7.2, we’ll need to follow these minimum steps:

1. Create a new Java class for the mob
2. Register the new mob class in the main mod
3. Give the mob a name
4. Create or assign a texture to the mob

We’ll walk through steps 1-3 in this lesson, so follow along yourself in Eclipse and reproduce our examples. Then in the lesson activity you’ll apply a texture to your new mob.

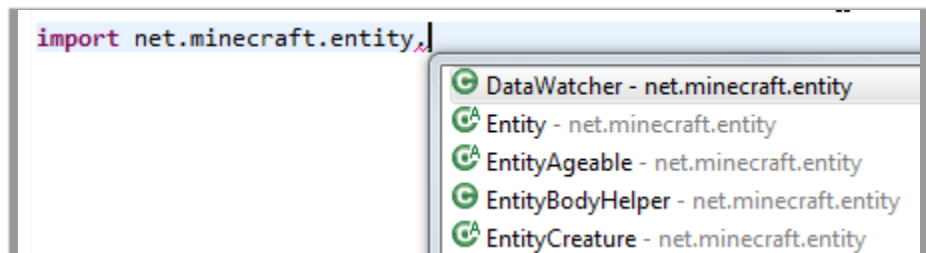
Creating an Entity

Using Eclipse, create a new Java class in your existing “**com.example.examplemod**” package. We’ll call our new mob “**BadGuyEntity.java**”. Here is your new class at the beginning:

```
package com.example.examplemod;

public class BadGuyEntity
{
}
```

Now, right away we need to make this class a type of Minecraft Entity. There are a variety of pre-defined entity types, and you can find them all by typing “import net.minecraft.entity.” and letting Eclipse show you some options.



We are going to make a monster, so we’ll import “net.minecraft.entity.monster.EntityMob”. Let’s add a few things to our **BadGuyEntity** class:

```
package com.example.examplerod;

import net.minecraft.entity.monster.EntityMob;
import net.minecraft.world.World;

public class BadGuyEntity extends EntityMob
{
    static final String name = "BadGuyEntity";

    public BadGuyEntity(World ourWorld)
    {
        super(ourWorld);
    }
}
```

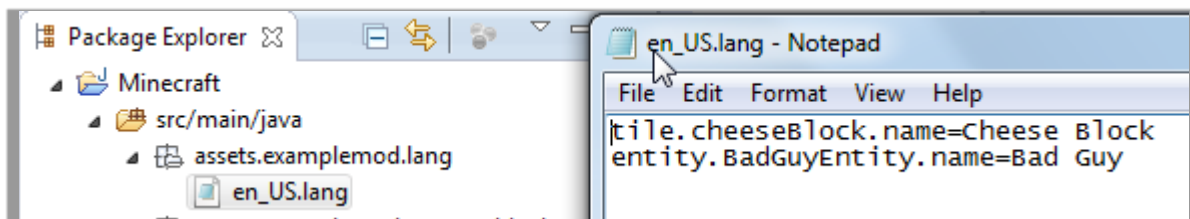
We have made our **BadGuyEntity** object extend the built-in **EntityMob**, so it will automatically behave like a generic monster in Minecraft. We also added some required **import** statements and a **String** name that will uniquely refer to our mob. Finally, we added a constructor function that simply accepts the **World** object and passes that to the base class constructor by calling the **super()** function. Now we have a very simple, but complete mob object!

Naming an Entity

Remember when we added a cheese block, we needed to add an entry to a language file so we could make a nice display name for the new object. Mobs are no different, so find your existing “en_US.lang” text file in your “src/main/java/assets/examplemod.lang” directory. Add a new entry as shown below:

```
tile.cheeseBlock.name=Cheese Block  
entity.BadGuyEntity.name=Bad Guy
```

Remember we are using the internal string “BadGuyEntity” to refer to our mob, so this line simply says that the entity with the string “BadGuyEntity” will have a name equal to “Bad Guy”. When you are done, your new file should look like the example below.



Registering an Entity

Finally, we need to tell our main **ExampleMod** object that we have a new mob! Let's return to our "ExampleMod.java" file and find the familiar **preInit()** function. Towards the bottom, after all of the other work you've already done for custom blocks, let's add some registration lines.

```
public void preInit(FMLInitializationEvent event)
{
    // other custom block code omitted

    // add our BadGuyEntity mob to the world
    int entityID = EntityRegistry.findGlobalUniqueEntityId();

    EntityRegistry.registerGlobalEntityID(BadGuyEntity.class,
        BadGuyEntity.name, entityID);

    EntityRegistry.registerModEntity(BadGuyEntity.class,
        BadGuyEntity.name, entityID, this, 64, 1, true);

    EntityList.entityEggs.put(Integer.valueOf(entityID),
        new EntityList.EntityEggInfo(entityID,
            Color.WHITE.getRGB(), Color.GREEN.getRGB()));
}
```

The first line creates a new, unique entity ID integer for our mob. Next we call **EntityRegistry.registerGlobalEntityID()**, passing in our new mob class, the string name, and the entity ID integer. This will link together those three things.

The third line calls **EntityRegistry.registerModEntity()** to associate the new mob class, name, and entity ID with some other parameters. The other parameters should be "this" (a reference to the main **ExampleMod** object), a tracking range, update frequency, and flag for velocity information packets. The values we've given, **64**, **1**, and **true**, work well for these parameters.

Finally, we want to call **EntityList.entityEggs.put()** to add a new **EntityEgg** to the game. Eggs are used to spawn (create) new monsters, so here we are creating a new **EntityEggInfo** object with the entity ID and two colors, white and green, for our egg.

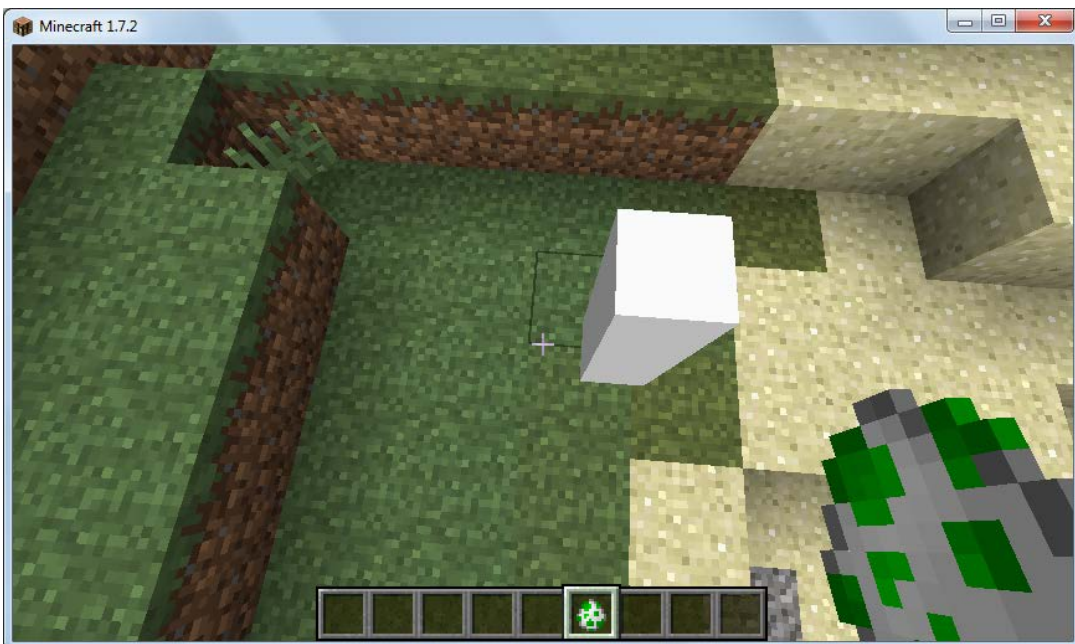
There are several new objects here such as **EntityList** that may be red-underlined, so use Eclipse to add the correct **import** statements for each object!

Testing

OK, your Minecraft project should now compile without any errors. Let’s test it out! When you build and run your game client, create a new world in Creative mode. Then press “E” and click over to the Miscellaneous tab. On this tab you should find the spawn egg for our new Bad Guy object, and it will be colored white and green. Put one of those eggs into your inventory.



Now you can select that egg back on the main screen, and right-click while pointing towards the ground. You should see a new “Bad Guy” object appear! We haven’t assigned a texture to this object, so it’s just a white rectangular solid at the moment. But you should see the object start to hop and move around just like a regular monster.



Further Reading

There are many, many more things you can do with a mob. You can create different types of mobs, add some custom behavior, and more. We can't cover it all, so you'll have to read more on your own and practice your Java skills to implement some of the more complex behavior.

Here are some links to mob-related tutorials. Some of them are for 1.6.X, while others bridge the gap with some 1.7.X information.

http://minecraft.gamepedia.com/Mods/Creating_mods/Creating_a_mob

<http://www.wuppy29.com/minecraft/modding-tutorials/wuppys-minecraft-forge-modding-tutorials-for-1-6-entity-part-1-registry/#sthash.rsWRTdsg.dpbs>

http://www.minecraftforge.net/wiki/Mob_Tut_%281.6.2%29

<http://www.wuppy29.com/minecraft/modding-tutorials/wuppys-minecraft-forge-modding-tutorials-for-1-6-entity-part-2-entitytutorial/#sthash.JLW1WhJx.dpuf>

<http://www.minecraftforum.net/topic/2389683-172-forge-add-new-block-item-entity-ai-creative-tab-language-localization-block-textures-side-textures/>

Of course, you can also search YouTube to find many related videos.

Learning good Java programming skills will be very important as you try to make more complex mods. If you are interested in self-study Java material, please see our [TeenCoder: Java Programming](#) course.

Lesson 8 Activity: Adding an Entity Texture

In this activity you are going to assign a texture to your new **BadGuyEntity** so he no longer appears as a white rectangle.

There are three steps for this process:

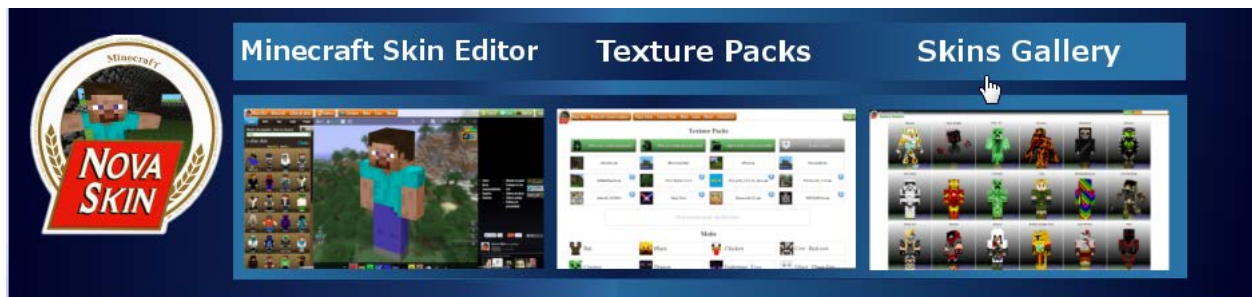
1. Create or find a texture image file to use
2. Create a new Java object to render (draw) the texture
3. Register that new render object in the main mod class

Creating or Finding Texture Images

Texture image files are in PNG format, and contain a series of tiles or images packed together in a single image. You usually don't want to edit this file by hand, so various people have created texture image editors to make life easier. The one we used for this lesson is called "Nova Skin" and can be found here:

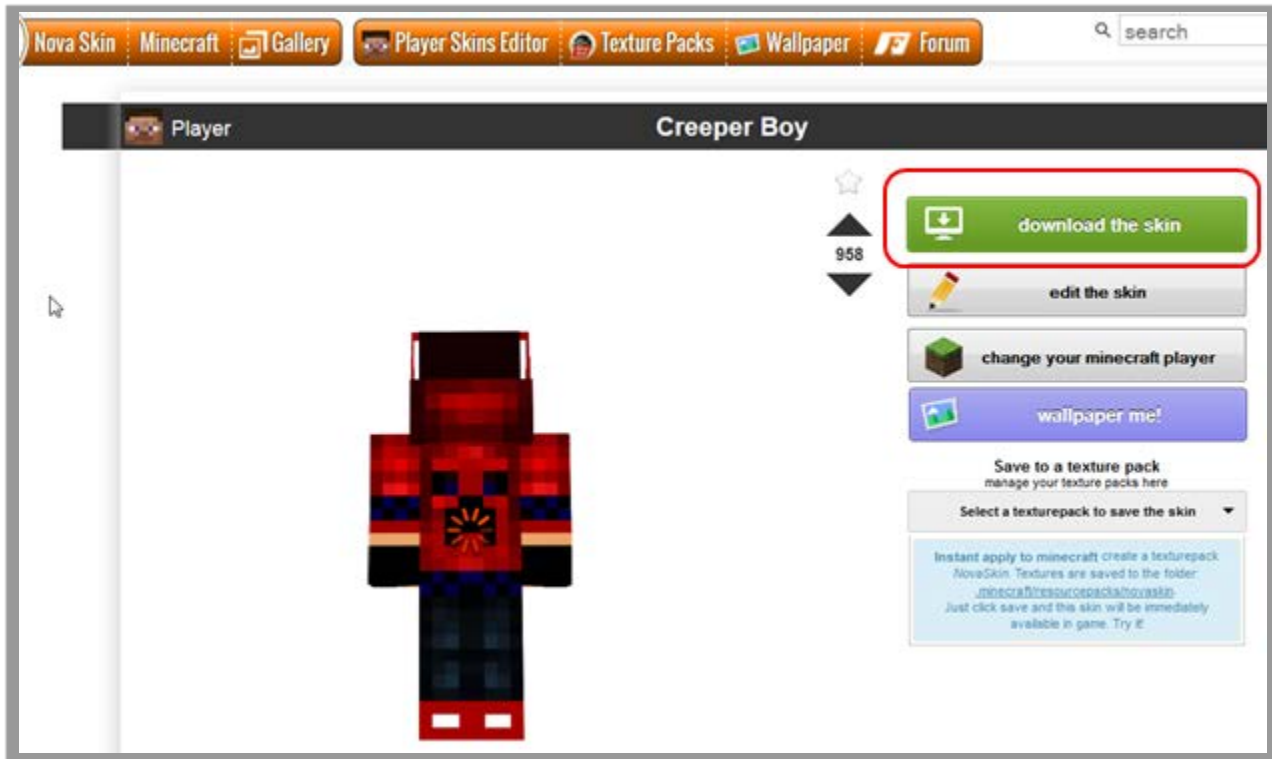
<http://minecraft.novaskin.me>

Across the top you'll see several options.



You can use the Skin Editor to create your own custom texture, or browse through the Skins Gallery to select a variety of pre-created textures. We're going to just pick something from the Gallery, and leave the Editor for your own exploration. Please note that the Gallery contains fan-inspired images, and a few of them may be on the offensive side, so steer clear of those.

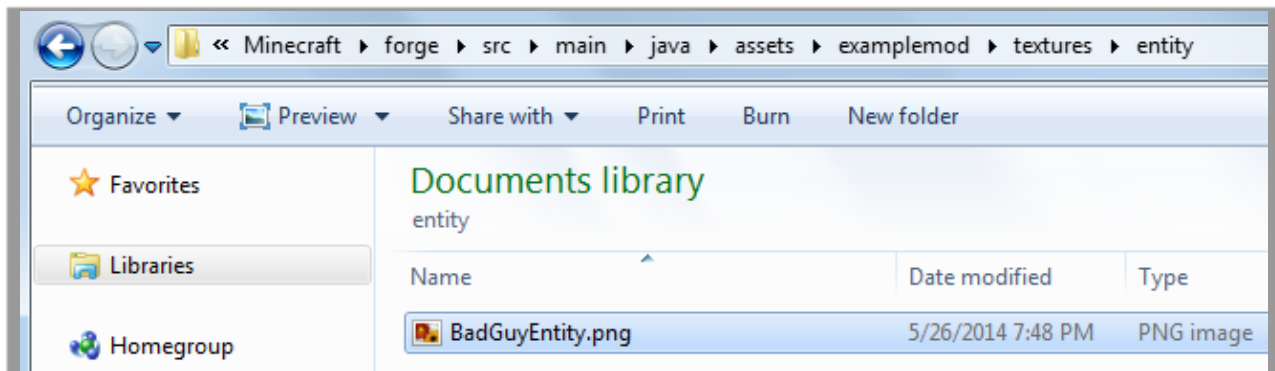
You can choose your own image, but we picked one from the gallery called “CreeperBoy”.



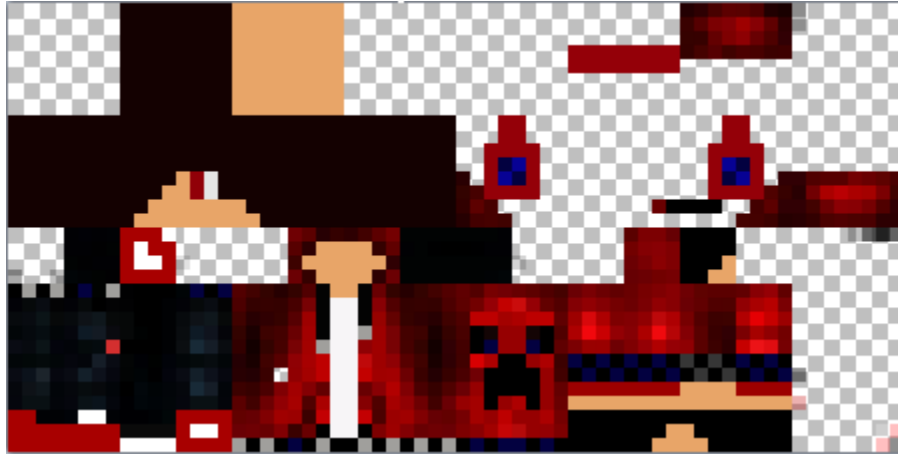
Your entity texture will go in a similar location to your custom Cheese block, but you’ll want to create an “entity” sub-directory instead of using “blocks”. Here is your complete target path.

```
“Forge/src/main/java/assets/examplemod/textures/entity”
```

Once you’ve selected your texture, click “download the skin. Save the file to your hard drive in the directory listed above, and change the filename to “BadGuyEntity.png”.

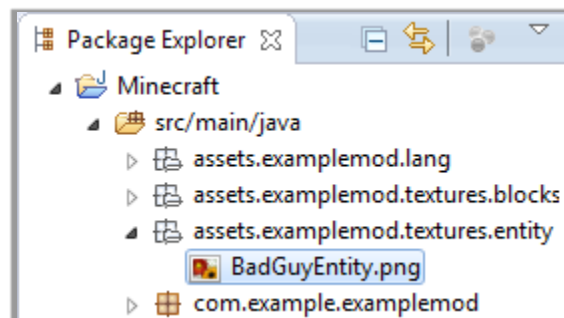


If you want to look inside this PNG file, you'll see some perplexing graphics similar to the ones below. This single file actually contains a number of sub-sections or tiles that will be broken apart to form the head, arms, and other body parts at run-time.



If your image file has a different format (perhaps more square-ish dimensions), then it might not be the right version for the mob we want to create. When you run Minecraft with an incorrect texture file, you may see exceptions on start-up, or just a random jumble of image junk on your mob. That's your sign to make sure the file is in the right location and format!

If you switch back over to Eclipse and select your "src/main/java" folder, and hit "F5" to refresh the view, you should see your new assets package and PNG file appear.



Rendering the Texture Image

We need a new Java class to tell Forge about our custom texture. Using Eclipse, create a new Java object called “RenderBadGuyEntity.java”, and then type in or cut-n-paste the full code shown below.

```
package com.example.exemplemod;

import net.minecraft.client.model.ModelBiped;
import net.minecraft.client.renderer.entity.RenderBiped;
import net.minecraft.entity.Entity;
import net.minecraft.util.ResourceLocation;

public class RenderBadGuyEntity extends RenderBiped
{
    private static final ResourceLocation badGuyTexture =
        new ResourceLocation("exemplemod:textures/entity/BadGuyEntity.png");

    public RenderBadGuyEntity(ModelBiped model, float f)
    {
        super(model, f);
    }

    @Override
    protected ResourceLocation getEntityTexture(Entity e)
    {
        return badGuyTexture;
    }
}
```

This class extends the **RenderBiped** class to make a humanoid mob. It then creates a **ResourceLocation** object and initializes it with the path to the “BadGuyEntity.png” we just copied into your new directory. It needs a constructor function that calls the **super()** base class with a couple of parameters. It also has a **getEntityTexture()** function that returns the **ResourceLocation** belonging to this object. So, any time this render object is used, anyone that asks for an image will be given the “BadGuyEntity.png” texture.

Make sure this class compiles without errors before you continue.

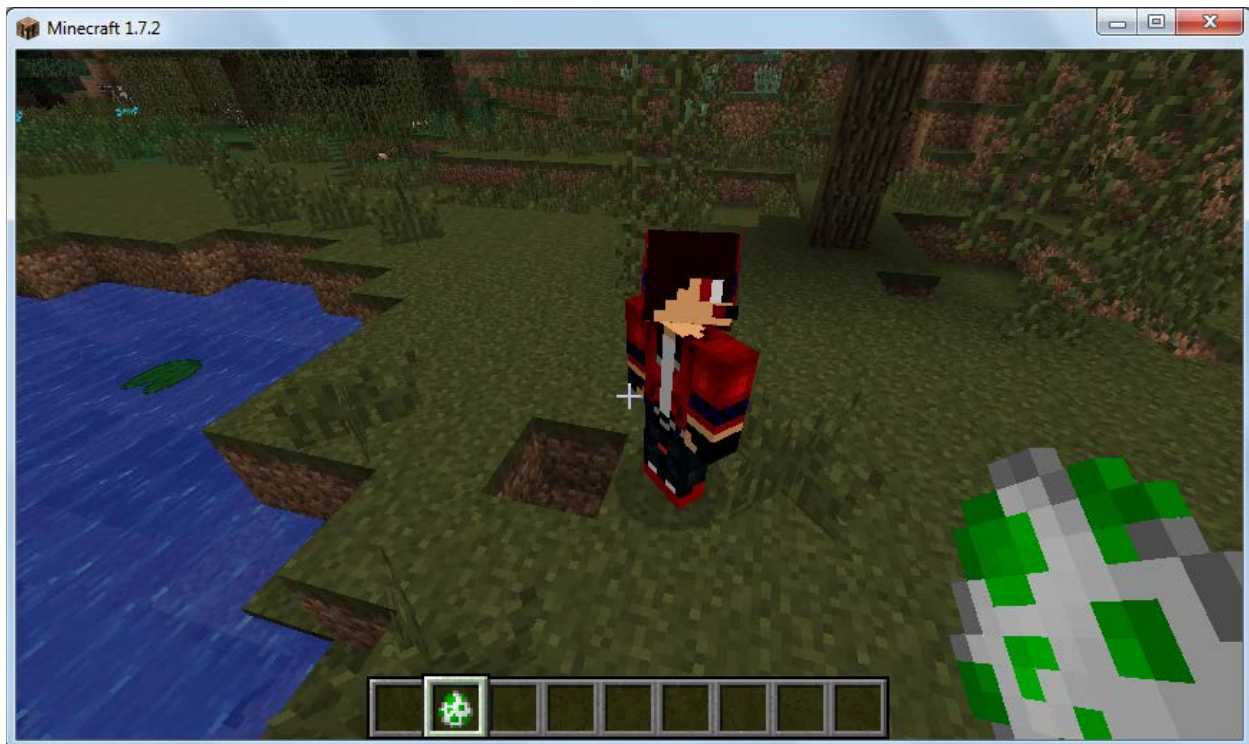
Registering the Render Object

We need to make one last change to our main “ExampleMod.java” file to register the new renderer. At the bottom of the `preInit()` function, under the registration of the `EntityEggInfo`, add this line:

```
RenderingRegistry.registerEntityRenderingHandler(BadGuyEntity.class,  
        new RenderBadGuyEntity(new ModelBiped(), 0.5F));
```

The `RenderingRegistry.registerEntityRenderingHandler()` function (say that 3 times fast) will associate the new `BadGuyEntity` class to a new copy of our `RenderBadGuyEntity` object. That render object will be created with a new `ModelBiped()` model and a floating point value (0.5F) to set the shadow size.

OK, you should be able to build and run everything without errors. When you return to your Minecraft world, your custom mob should now have a great-looking texture!



Feel free to experiment with different models, textures, and other properties on your own. The best source of information and examples, as always, is your careful online search to find people that have already solved your particular problem.

Activity Solution

Here is the full solution code for this activity.

BadGuyEntity.java:

```
package com.example.exemplemod;

import net.minecraft.entity.monster.EntityMob;
import net.minecraft.world.World;

public class BadGuyEntity extends EntityMob
{
    static final String name = "BadGuyEntity";

    public BadGuyEntity(World ourWorld)
    {
        super(ourWorld);
    }
}
```

RenderBadGuyEntity.java:

```
package com.example.exemplemod;

import net.minecraft.client.model.ModelBiped;
import net.minecraft.client.renderer.entity.RenderBiped;
import net.minecraft.entity.Entity;
import net.minecraft.util.ResourceLocation;

public class RenderBadGuyEntity extends RenderBiped
{
    private static final ResourceLocation badGuyTexture =
        new ResourceLocation("exemplemod:textures/entity/BadGuyEntity.png");
}
```

```
public RenderBadGuyEntity(ModelBiped model, float f)
{
    super(model, f);
}

@Override
protected ResourceLocation getEntityTexture(Entity e)
{
    return badGuyTexture;
}
}
```

ExampleMod.java:

```
package com.example.examplemod;

import java.awt.Color;

import net.minecraft.block.Block;
import net.minecraft.block.material.Material;
import net.minecraft.client.model.ModelBiped;
import net.minecraft.creativetab.CreativeTabs;
import net.minecraft.entity.EntityList;
import net.minecraft.init.Blocks;
import net.minecraft.init.Items;
import net.minecraft.item.ItemStack;
import cpw.mods.fml.client.registry.RenderingRegistry;
import cpw.mods.fml.common.Mod;
import cpw.mods.fml.common.Mod.EventHandler;
import cpw.mods.fml.common.event.FMLInitializationEvent;
import cpw.mods.fml.common.registry.EntityRegistry;
import cpw.mods.fml.common.registry.GameRegistry;

@Mod(modid = ExampleMod.MODID, version = ExampleMod.VERSION)
```

```
public class ExampleMod
{
    public static final String MODID = "examplemod";
    public static final String VERSION = "1.0";

    public static final Block cheeseBlock = new CheeseBlock(Material.ground);

    @EventHandler
    public void preInit(FMLInitializationEvent event)
    {
        // some example code
        GameRegistry.registerBlock(cheeseBlock, "cheeseBlock");
        GameRegistry.registerWorldGenerator(new MyWorldGenerator(),0);

        // shapeless recipe to create diamonds from cheese
        ItemStack inputStack = new ItemStack(cheeseBlock);
        ItemStack outputStack = new ItemStack(Items.diamond);
        GameRegistry.addShapelessRecipe(outputStack, inputStack);

        // shapeless recipe to create a baked potato from 2 cheese and 1 potato
        ItemStack cheeseStack = new ItemStack(cheeseBlock);
        ItemStack potatoStack = new ItemStack(Items.potato);
        ItemStack bakedStack = new ItemStack(Items.baked_potato);
        GameRegistry.addShapelessRecipe(bakedStack,
                                         cheeseStack, cheeseStack, potatoStack);

        // shaped recipe to create a baked potato from 2 cheese and 1 potato
        ItemStack cheeseStack2 = new ItemStack(cheeseBlock);
        ItemStack potatoStack2 = new ItemStack(Items.potato);
        ItemStack bakedStack2 = new ItemStack(Items.baked_potato);
        GameRegistry.addRecipe(bakedStack2, "x ", " y ", " x",
                               'x', cheeseStack2, 'y', potatoStack2);
    }
}
```

```
// shaped recipe to create diamonds from cheese and potatoes and coal
ItemStack coalStack = new ItemStack(Items.coal);
GameRegistry.addRecipe(new ItemStack(Items.diamond, 10),
    "x ", " y ", "zzz",
    'x',cheeseStack, 'y', potatoStack,'z',coalStack);

// smelting recipe to create diamonds from bricks
ItemStack diamondStack = new ItemStack(Items.diamond);
GameRegistry.addSmelting(Items.brick, diamondStack, 0.5F);

System.out.println("CHEESE BLOCK >> " + cheeseBlock.getUnlocalizedName());

// add our BadGuyEntity mob to the world
int entityID = EntityRegistry.findGlobalUniqueEntityId();

EntityRegistry.registerGlobalEntityID(BadGuyEntity.class,
    BadGuyEntity.name, entityID);

EntityRegistry.registerModEntity(BadGuyEntity.class,
    BadGuyEntity.name, entityID, this, 64, 1, true);

EntityList.entityEggs.put(Integer.valueOf(entityID),
    new EntityList.EntityEggInfo(entityID,
    Color.WHITE.getRGB(), Color.GREEN.getRGB()));

RenderingRegistry.registerEntityRenderingHandler(BadGuyEntity.class,
    new RenderBadGuyEntity(new ModelBiped(), 0.5F));
}
}
```

en_US.lang:

```
tile.cheeseBlock.name=Cheese Block
entity.BadGuyEntity.name=Bad Guy
```


WHAT'S NEXT?

Congratulations, you have finished Part Two of our free lesson series on *Beginning Minecraft Mods*! We may create additional lessons in the future, so check back with us periodically.

If you are enthusiastic about programming and would like to have more formal training in Java, C#, Visual Basic, or HTML, you can find award-winning, self-study courses at our website.

Course	Description
KidCoder: Web Design	Use HTML, CSS, and JavaScript to create web pages
KidCoder: Visual Basic	Learn Visual Basic to write your own programs and games
TeenCoder: C#	Use object-oriented concepts to write programs and games in C#
TeenCoder: Java/Android	Learn Java and Eclipse, study for the AP CS A exam, and write Android apps!

We hope you have enjoyed this experience, and want to continue creating Minecraft mods on your own. There are many great free resources on the Internet to learn more about Minecraft mods. Just do a bit of searching online or on YouTube to see what you can find!

We welcome your feedback on these lessons and suggestions for future topics. You can find us online at:

<http://www.homeschoolprogramming.com>

You can also follow us on Facebook and Twitter:



<http://www.facebook.com/HomeschoolProgramming>



<http://twitter.com/HSProgramming>