ZedCraft:

Spigot Plugin Setup Using JetBrains IntelliJ (v1.0)

CPRE 488 Team S1-1

Simon Aguilar, Trevor Friedl, Ricky Smith, Justin Templeton

(Revised and completed for CPRE 488 Spring 2024)

Table of Contents:

Part 1 - Introduction:

In this (document), we will explain the required and recommended processes to use to achieve the following objectives:

- Install and configure JetBrains IntelliJ Community Edition on the development PC
- Create or clone IntelliJ "Spigot" project, intended for Minecraft plugin development
- Learn how to use Spigot API to call in-game functionality using Java logic
- Learn how to build and upload plugins into the Minecraft server file directory

This (document) is intended for top-down reading but can be re-referenced when specific issues arise throughout the following process. We want to express and disclaim that the methods used within this (document) ARE NOT proven to be the only or best solutions to these cases. Through thorough research and trial, we have found the methods listed below reliable and relatively straightforward for the uses within the ZedCraft project. Given further updates in the future regarding the project and with everchanging versions of each of the systems, we anticipate a potential need to change the exact process to replicate the below processes. In all, future improvements to the project may require updates to the documentation.

In this (document), we will discuss all the processes required to get IntelliJ installed and adequately set for our Spigot plugin and the different steps best to understand the software development lifecycle within this process. We will discuss how to import the ZedCraft plugin project into IntelliJ, and we will discuss the different tools built-in that will aid and help in the plugin development process so that we can deploy the JAR file used for the plugin we want to create. This (document) will also include mentioning common pitfalls and techniques used by IntelliJ as efficiently as possible.

Part 2 - JetBrains IntelliJ Setup:

JetBrains is a vendor of several software development tools specifically intended to accelerate and enable the production of more efficient and error-proof code. IntelliJ is JetBrains's Java/Kotlin-focused IDE distribution, similar to Oracle's Eclipse. One of the pros to using IntelliJ is that lots of the necessary tools beyond the Java JDK come installed with the IDE, such as build tools (Maven, Gradle, etc.) and different project support types (Android Studio, Minecraft development, etc.). Fortunately, IntelliJ supports development for Minecraft out-of-the-box, which makes IntelliJ our believed best choice for developing plugin applications for the ZedCraft Project.

First, we will install IntelliJ Community Edition from the JetBrains website. The link to this download is found here: https://www.jetbrains.com/idea/download/?section=windows. **Scroll down and download the Community Edition, not the Ultimate Edition.**

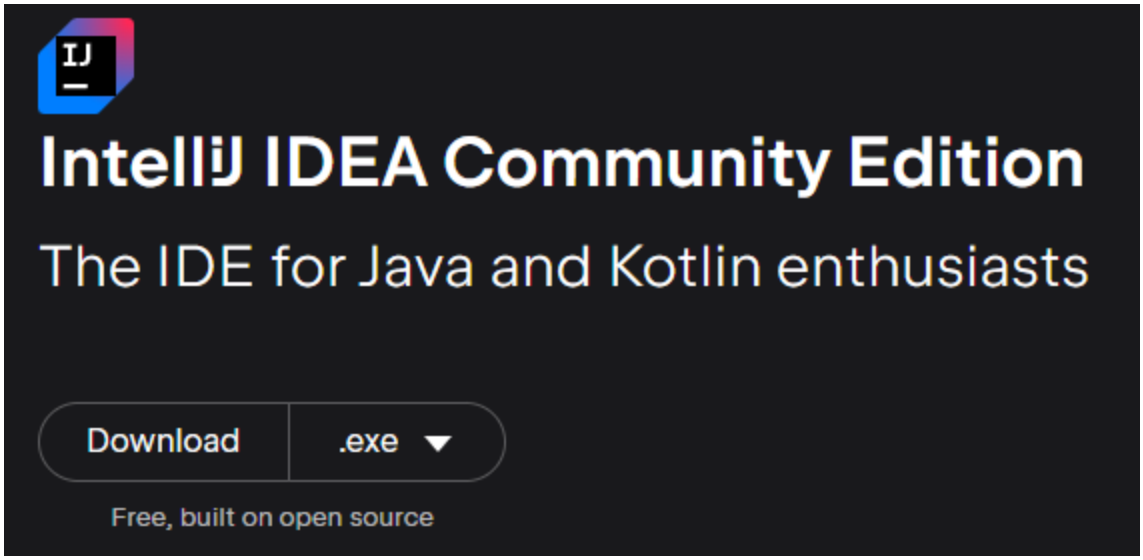

Figure 1 - IntelliJ IDEA Download Icon (May 2024)

Once downloaded, the install wizard will walk you through adding any additions one might be interested in supporting. Figure 2 below details some suggested options to select when installing for the first time.

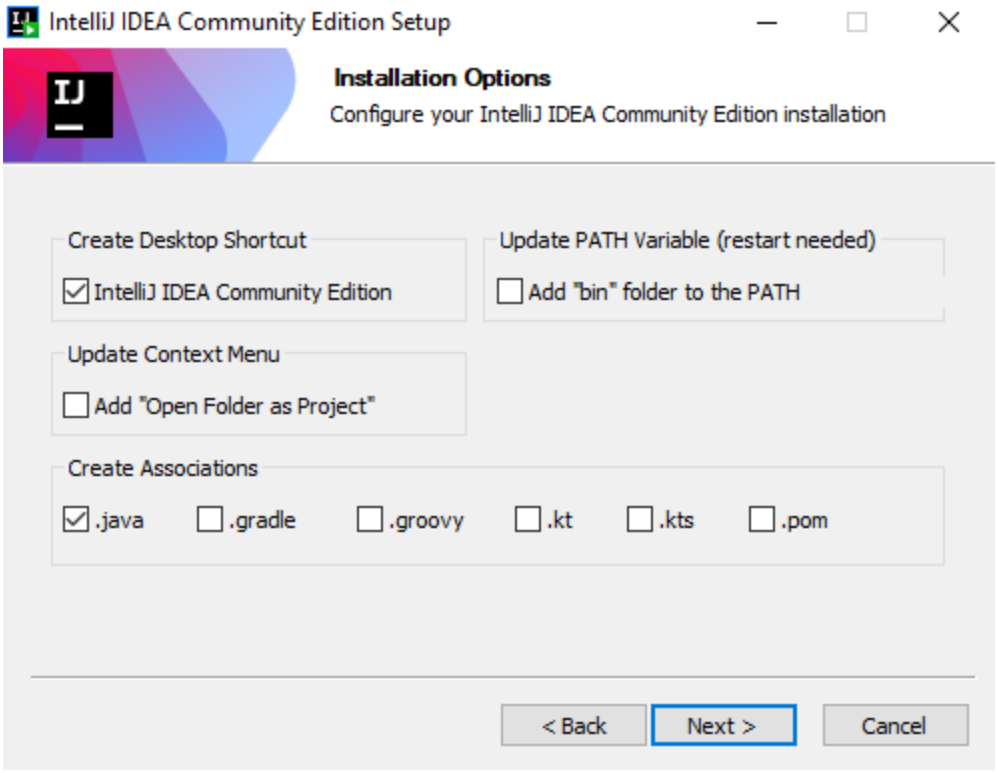

Figure 2 - Installation Options

Beyond this point, IntelliJ will do the rest of the heavy lifting to install the application. Once everything is complete, press finish, launch IntelliJ, and you should be good to go.

Part 3 - Importing the ZedCraft Java Plugin:

Once you've installed IntelliJ onto your development PC, the next step is to get the ZedCraft plugin cloned from the git repo and imported as a project into the IDE. The first step will be to visit the ZedCraft GitHub repository currently hosted at https://github.com/TJFriedl/ZedCraft. Using your preferred method, clone the repository onto your development PC. First, ensure that you have git installed on your development PC by running "git" in a command prompt on your PC. If you get a prompt for git, then you should have git available for use. Once this has been checked, run the following command on your PC (unless using an SSH key is preferred):

"git clone https://github.com/TJFriedl/ZedCraft.git"

Once you've run the command, confirm that the contents of the ZedCraft project are available in a directory on your PC. Also, ensure that all the estimated contents of the repository are available to you, mainly the *"plugin/"* directory, as this is the main package we will work with for the Spigot plugin development. Now, navigate back to the IntelliJ application and launch it. You should be introduced to a welcome screen; click the "Open" option, where we will be prompted to open an existing IntelliJ project. Luckily, the *"plugin/"* directory is already pre-configured to an IntelliJ project within the git, so importing it should be straightforward. Go ahead and navigate to the *"ZedCraftPlugin"* directory in IntelliJ and press OK, as shown in Figure 3 below.

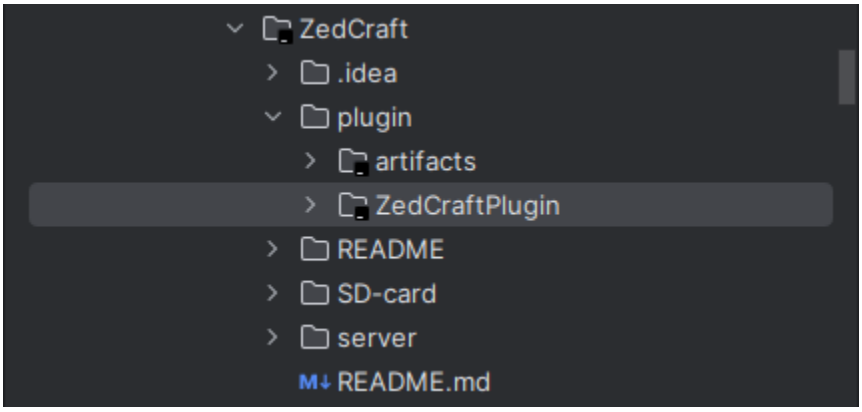

Figure 3 - Directory Tree for ZedCraftPlugin

The ZedCraftPlugin project should open and be pre-configured as a Spigot plugin project upon pressing OK. If the IDE prompts you to, install the Minecraft Development Package IDE plugin and give it a proper JDK for indexing and compiling. **Make sure to use a JDK version of at least 1.8; otherwise, Minecraft will not compile correctly**.

At this point, feel free to navigate through the contents of the plugin source code. Relative to the potential size of other sources of Spigot plugins, ZedCraft has a fair amount of content. Make sure to attend to any errors the IDE might be throwing, and refer to the documentation or online material to best fix these issues before moving forward or further working on the plugin.

Part 4 - Understanding the Spigot API:

As currently configured, the ZedCraft spigot plugin currently uses two external dependents for the build of the plugin, the Spigot 1.8 API and the javax.json API (used for some of the JSON file parsers in the plugin). The version of the Spigot API is specific to mirror the version of the Minecraft server and client that is being used to run and use the plugin. Project configuration for the version of the Spigot API can be found in the *"pom.xml"* file found in the project directory, which is used in the configuration of the Maven project used to help build the target content and JAR executable of the plugin. Yes, it is possible to update the version of the Spigot API directly from within the project. You only need to change the *"<version>"* entry to fit the new version you want to interface. After changing this, you must ensure the server and client run your new JAR's destined version. Figure 4 below details the entry for the Spigot plugin dependency in the *"pom.xml"* file.

```
<repositories>
    <repository>
        <id>spigotmc-repo</id>
        <url>https://hub.spigotmc.org/nexus/content/repositories/snapshots/</url>
    </repository>
    <repository>
        <id>sonatype</id>
        <url>https://oss.sonatype.org/content/groups/public/</url>
    </repository>
</repositories>

<dependencies>
    <dependency>
        <groupId>org.spigotmc</groupId>
        <artifactId>spigot-api</artifactId>
        <version>1.8-R0.1-SNAPSHOT</version>
        <scope>provided</scope>
    </dependency>
```

Figure 4 - ZedCraftPlugin Project pom.xml entries required for Spigot 1.8 API

The Spigot API is a layer of classes targeting clean serialization and encapsulation of Minecraft packet calls, building on top of NBT (Name Binary Tag, Minecraft-specific) packets used, interfacing data transfer from server to game, and vice-versa. Spigot is a fork of the *"Bukkit API"* intended to supersede its functionalities as one of the most previously popular server API interfaces.

On the surface, the Spigot API can be overwhelming and challenging to understand- especially to a programmer unfamiliar with the technology and functionality of Minecraft. Luckily, Spigot provides Javadoc-style documentation for each version of their API to list all of the API function calls. Documentation for the Spigot 1.8 API can be found here: https://helpch.at/docs/1.8/overview-summary.html.

The main branches of functionality implemented within the ZedCraftPlugin are currently listed as command executors and event listeners. Command executors are responsible for registering and deploying logic for customized in-game commands and capturing and editing game-specific events' logic. For example, the server can trigger an event captured upon a player joining the server and send them a customized message, as seen by the provided code in Figure 5.

```
public class PlayerJoinEvent implements Listener {  2 usages

    // Example event handler for player join event
    @EventHandler
    public void onJoin(org.bukkit.event.player.PlayerJoinEvent e) {
        e.getPlayer().sendMessage( s: ChatColor.GOLD + "Welcome, You are playing Minecraft" +
                " on Java 1.8 on a ZedBoard FPGA.");
    }

}
```

Figure 5 - Code snippet for custom game event

The best way to become familiar with the Spigot API is to practice and read documentation and examples provided online and access other media, such as blogs and YouTube, to seek walkthrough examples on exploring and experimenting with specific functionalities.

Another good portion of the Spigot plugin environment is to study the functionality of the onEnable() and onDisable(), typically found within the main class. These methods require an override tag that allows the server to operate upon startup and shutdown. Figure 6 below details an example of the ZedCraft main class implementation for server startup and shutdown.

```java
public final class Main extends JavaPlugin implements Listener {  12 usages

    public static Main main;  6 usages
    public static HashMap<String, BlockData> blocks;  4 usages

    @Override  no usages
    public void onEnable() {
        main = this;
        getServer().getConsoleSender().sendMessage( s: ChatColor.GOLD + "ZedCraft Server starting up...");

        // Do everything we need to do in order for the plugin to be configured
        try {
            //Set up folder packages
            File directory = new File(this.getDataFolder().getPath());
            if (!directory.exists()) { directory.mkdirs(); }
            blocks = DataCollection.populateJSONMap();

            //Register in-game commands
            CameraCommands cc = new CameraCommands();
            getCommand( name: "picture").setExecutor(cc);
            getCommand( name: "giveblockdata").setExecutor(cc);

            //Register events down below
            this.getServer().getPluginManager().registerEvents(new PlayerJoinEvent(),  plugin: this);
        } catch (Exception e) {
            getServer().getConsoleSender().sendMessage( s: ChatColor.DARK_RED + "[ZedCraft] ERROR upon plugin startup");
            throw new RuntimeException(e);
        }

        getServer().getConsoleSender().sendMessage( s: ChatColor.GREEN + "[ZedCraft] Boot success!");
    }

    @Override  no usages
    public void onDisable() {
        // Plugin shutdown logic
        getServer().getConsoleSender().sendMessage( s: ChatColor.RED + "[ZedCraft] Server shutting down...");
    }
}
```

Figure 6 - Main class implementation for onEnable() and onDisable() method calls

Figure 6 details a couple of different operations in the startup phase. First, we initialize a try/catch case to ensure that the plugin's default data folder is created or otherwise instantiates it. Then, an arbitrary call is made to populate a HashMap variable "blocks" populated using a call from another class. Then, both the "picture" and "giveblockdata" commands are registered with a new "CameraCommands" command executor object. The plugin will print a boot success sequence using standard output if all tasks succeed. Upon shutdown, the server will send a shutdown message sequence.

Part 5 - Building the Plugin JAR File:

As you're either getting the plugin to a new state, you would like to deploy and test, or whether you are looking to get an initial build of the JAR file to import onto your server, it is essential to understand the process to build and acquire the JAR file for your plugin. As mentioned, the Minecraft server development add-on in the IntelliJ IDE uses Maven to build its projects, so we will use the different lifecycle functions to extract our target JAR file.

The first thing to do upon using the Maven project build tools for the first time is to make a clean build of the project. To do this, open up the maven sidebar on the side of the application, denoted by a button that provides an *"m,"* and press it to open it. You should now see a sidebar containing the different Maven build tools, as shown in Figure 7.

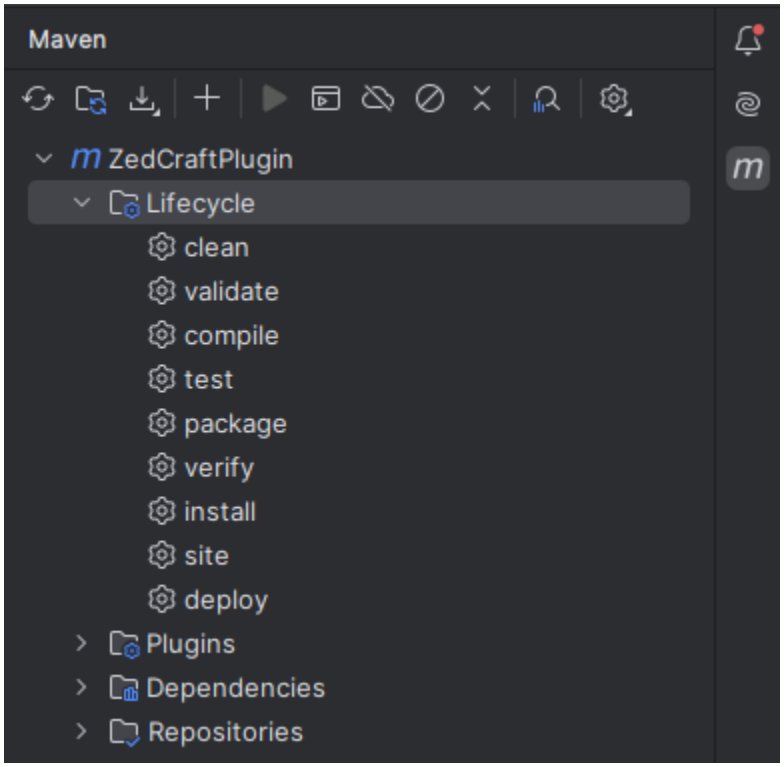

Figure 7 - Maven sidebar tools

Note that a "clean" task can be run within the lifecycle dropdown. To make a clean build of the project, double-click "clean" and let the IDE run the job. This will reset the project and delete the *"target/"* folder within the package. To build the project and the JAR plugin responsible for running the plugin on the server, click the "verify" job and notice that a new *"target/"* package will be generated in the explorer sidebar, as shown in Figure 8.

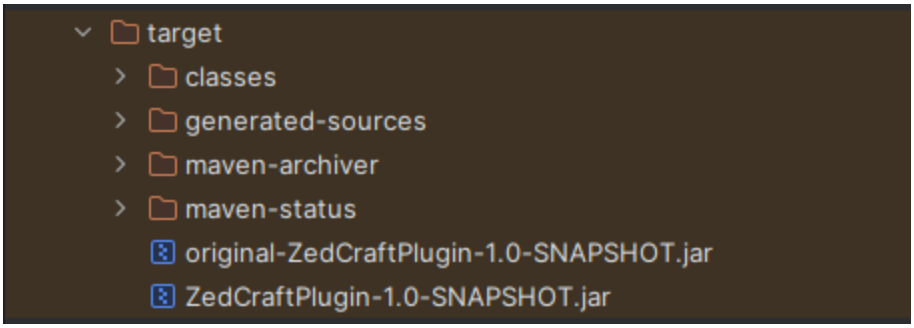

Figure 8 - New target directory, including plugin JAR build(s)

The generated JAR files are what the plugin will be built into- these are what the server needs to run the plugin correctly. Next, we will learn where to put this plugin to run and test it.

Part 6 - Attaching JAR File to Server:

Now that we have our plugin JAR artifact, we can import the JAR file into the server's *"plugins"* directory to get the plugin loaded onto the server. First, navigate the server directory you have created for testing and use (there should be a *"server/"* directory included in the ZedCraft repository- feel free to use this). Paste the JAR file into the plugins directory once you've navigated to "server/plugins/" inside the ZedCraft cloned repository. Now that the JAR file is inside the plugin's directory (and given no other substantial errors from the JAR build), the JAR file should be used upon startup of the server.

To test this, go back into the *"server/"* directory and run the following:

"java -Xmx256M -Xms256M -jar server.jar"

Once you've started receiving feedback from the server, you should see a message similar to the one seen in Figure 9. Congratulations if that is the case- you've successfully tied the plugin to the server!

```
[20:36:06 INFO]: [ZedCraft] Blocks HashMap Populated.

[20:36:06 INFO]: Boot success!
```

Figure 9 - Successful plugin boot message (as shown in Fig. 6 example)