Learn to Code in Minecraft Workshop
Miller Place Robotics Team 514

Day 2 - Writing your First Plugin

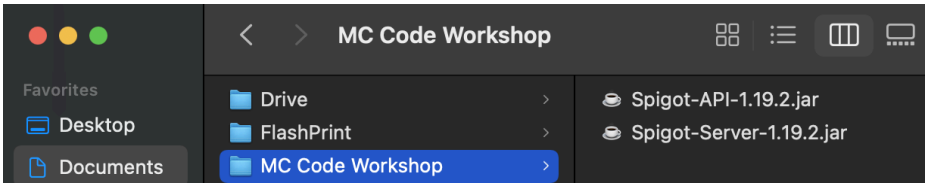
Objective

Today's goal is to create a new Java Project, write some simple code to create your plugin, and build the plugin to a JAR file. We briefly talked about it last week, but "plugins" are Java programs that can be used to add features to a Minecraft server. Spigot is what allows us to do that - they created a **framework** that helps us add (or plug-in) our code to a Minecraft server.

If time allows, you will also create a Spigot server on your laptop to test your plugin. If you complete these instructions ahead of the group, let us know and we'll get you the Spigot server instructions.

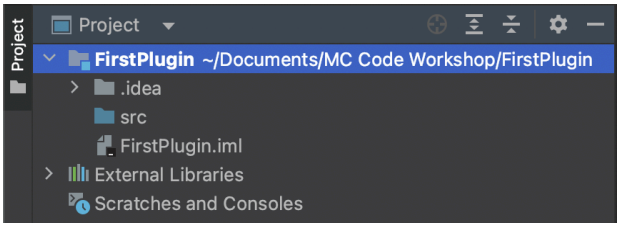
1 - Create a space on your laptop to store Workshop files

- Go to the **Documents** folder on your computer and create a folder called "MC Code Workshop". Having everyone store their files in the same folder makes it easier for us to help you if needed.
- We'll need a copy of the Spigot JARs from last week in this new folder. Try and find the two files from last week - **Spigot-API-1.19.2.jar** and **Spigot-Server-1.19.2.jar** - and copy them to the new folder you created. If you lost those files, ask for help.
- Make sure to leave a copy of the API jar in this folder from now on. When we reference it through our projects, IntelliJ won't be able to find it if it's moved or deleted.



2 - Create a new project in IntelliJ

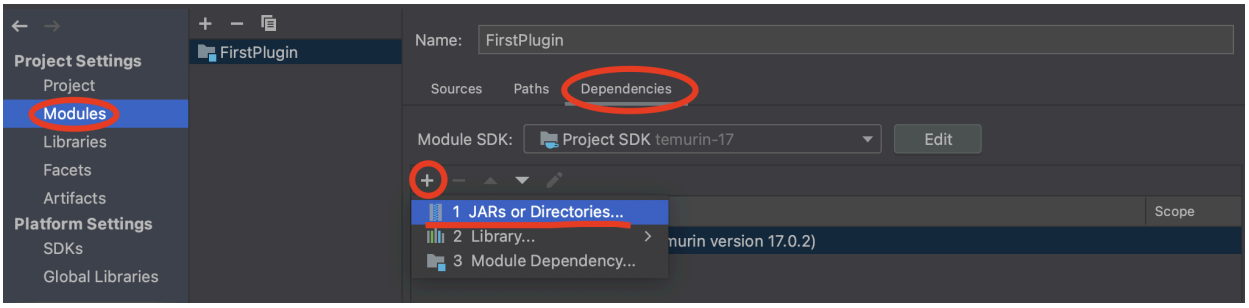
- From the IntelliJ main menu, click **New Project**. Name the project "FirstPlugin". Make sure the project is saved within the "MC Code Workshop" folder you created in Step 1.
- Uncheck "Create Git repository". Make sure **Language** is set to "Java", **Build System** is set to "IntelliJ", and **JDK** is set to the JDK you installed last week (should be named "temurin-17", "17", or something similar; ask if you're unsure). Don't change the Advanced Settings. Click **Create**.



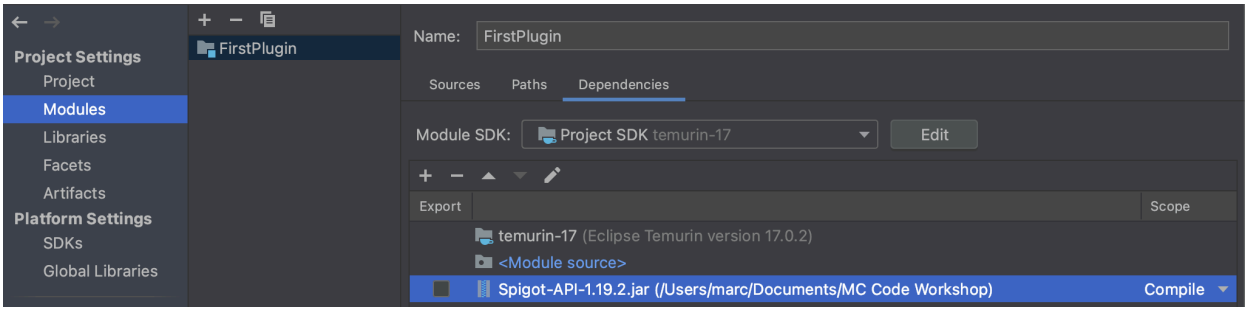
- On the left side of the window is the **Project menu**, where the project files are stored.
- The ".idea" folder and the "FirstPlugin.iml" file both help IntelliJ manage your project. You don't need to edit these.
- The "src" folder is standard for Java projects and contains the project's **Source Code**. This is where your Java files will go.
- **External Libraries** is a place where you can browse the code from libraries your project uses. A Code Library is a collection of code referenced by other projects. In our case, we will have two External Libraries - the JDK (Java Development Kit) we installed last week, and Spigot (a community-created Minecraft server program).
- Spigot isn't listed under External Libraries yet, though, since you need to add it.

3 - Add Spigot as an Artifact in your new Project

- Last week, we added Spigot-API-1.19.2.jar as a **Dependency** to the ExampleProject everyone downloaded. We need to do the same here so you can use Spigot to build your plugin.
- Open the Project Structure menu (File -> Project Structure), and go to the **Modules** tab. Within the "FirstPlugin" module, go to the **Dependencies** tab. Click the + sign, and click the first option.



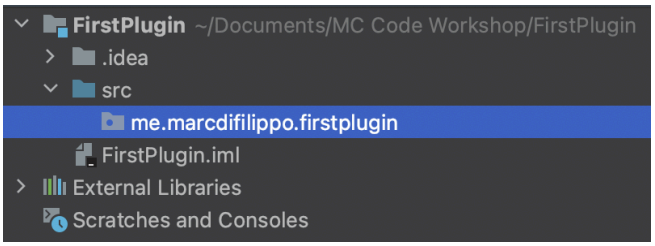
- Remember back in Step 1 we saved a copy of **Spigot-API-1.19.2.jar** in your folder for this workshop? That's the one you want to link here!



- All set now! Click Apply and OK. You should now see Spigot under "External Libraries".

4 - Creating a package for your code

- With the project setup complete, we can start writing code! We need to create a Package to store your code in. Java Packages help keep code organized. Here are some examples:
  - com.google.gson
  - org.bukkit.entity
- You'll notice that Packages look like reverse website addresses - google.com becomes com.google. For your project, follow this naming scheme: me.[your name].[plugin name]. For example, me.marcdifilippo.firstplugin. Keep in mind - packages should be all lowercase!
- To create your Java Package, right click on **src** and select "New -> Package", then type your package name and hit Enter.
- After that, you should have a Java Package. All your Java classes will be created here.



5 - Creating the plugin's Main Java class

- In Java, a Class is the place you will write Java code. Each Class gets its own file. For example, a Class named "Main" would be saved in a file named "Main.java". Classes can be a lot more complicated than this, but we'll skip that for now.
- Right-click on your Package and select "New -> Java Class". Call it Main, and hit Enter. You should see something like the code shown to the right ->
- The first line defines the package the Class exists in. Don't change this unless needed!

```
1  package me.marcdifilippo.firstplugin;
2
3  public class Main {
4  }
5
```

- Line 3 defines a **public** class named Main. **public** defines what other Java Classes can reference this class. Java resources can be **public**, **private**, or **protected**. We'll explain what this means later. You want your Main class to be **public** so Spigot's classes can use it.
- The curly braces on lines 3 and 4 are the boundaries of the Main class. Any code you want in the Main class **must** be between those two curly braces.
- We have one change to make, though! We want to add "extends JavaPlugin" between Main and the left curly brace, like shown to the right ->
- Also, if IntelliJ doesn't add it for you, you'll need to add the **import** line shown on line 3. This tells the JDK what specific package.class you want to use when saying **JavaPlugin**.

```
1  package me.marcdifilippo.firstplugin;
2
3  import org.bukkit.plugin.java.JavaPlugin;
4
5  public class Main extends JavaPlugin {
6  }
```

6 - Create the onEnable and onDisable methods

- Last week we briefly discussed **Frameworks** and how they can be helpful when you want to integrate your code in a larger code base. We use the WPILib Framework on our Robotics team, and here we will use the Spigot Framework.
- Spigot has a bunch of Java classes and methods that help us “plug in” our code to a Spigot server. We used one in the last step when we added “extends JavaPlugin” to the Main class definition!
- Now, we want to create **onEnable** and **onDisable** methods within the Main class. Spigot has pre-defined these methods in the **JavaPlugin** class, so it already knows about them. These methods get called when the server starts up and when it shuts down.

```
5   public class Main extends JavaPlugin {
6
7       @Override
8       public void onEnable() {
9
10      }
11
12      @Override
13      public void onDisable() {
14
15      }
16  }
```

- Any code we add within these methods gets run when the server starts and stops. Let’s add a line that prints a message to the server console.

```
8       @Override
9       public void onEnable() {
10          Bukkit.getLogger().info( msg: "My first plugin is starting up!");
11      }
12
13      @Override
14      public void onDisable() {
15          Bukkit.getLogger().info( msg: "My first plugin is shutting down!");
16      }
```

- Remember: You’ll need to import the Bukkit class so the JDK knows which class you’re referring to! You can have IntelliJ do this automatically as you type it out. If you type “Buk” and hit Enter to let IntelliJ autocomplete the class name, it will auto-import the class!

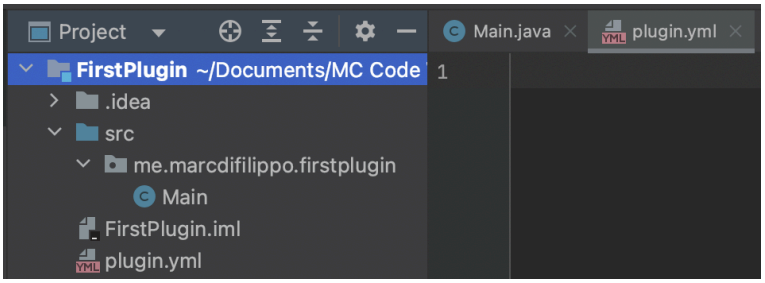
```
1   package me.marcdifilippo.firstplugin;
2
3   import org.bukkit.plugin.java.JavaPlugin;
4
5   public class Main extends JavaPlugin {
6
7       @Override
8       public void onEnable() {
9           Buk
10      }
11
        Bukkit org.bukkit
        BukkitCommand org.bukkit.command.defau
        BukkitRunnable org.bukkit.scheduler
```

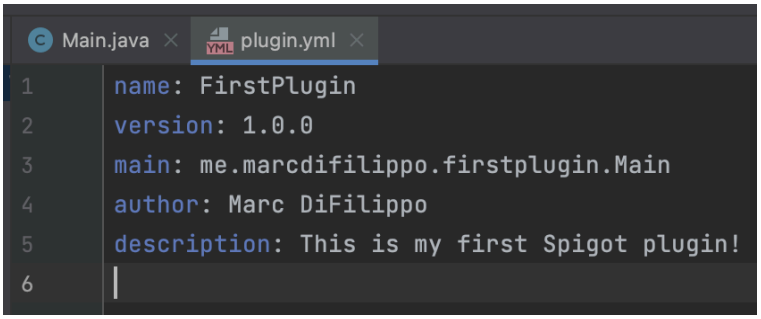
```
1   package me.marcdifilippo.firstplugin;
2
3   import org.bukkit.Bukkit;
4   import org.bukkit.plugin.java.JavaPlugin;
5
6   public class Main extends JavaPlugin {
7
8       @Override
9       public void onEnable() {
10          Bukkit
11      }
```

These next few steps will help you build your plugin to a Jar file. If time allows, we can cover more Java programming before the workshop ends. If not, we'll cover Commands and Listeners next week.

7 - Create the plugin.yml file

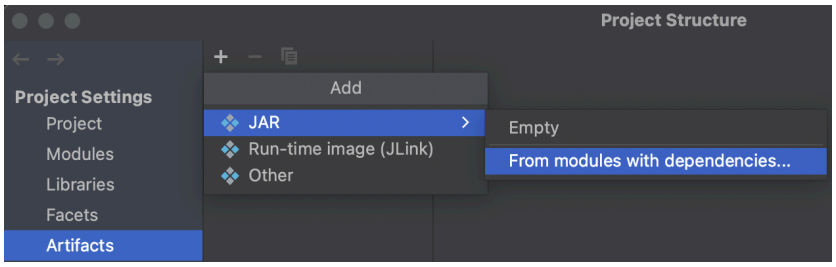
- In the **Project** menu, right-click on your project's name and select "New -> File". Name it **plugin.yml** and hit Enter to create it.
- The plugin.yml file tells Spigot key information about your plugin, such as its name, version, and where your Main Class is. Add the information on the right to your plugin.yml file.
- Change the values to match your name, your plugin, etc. Make sure there are no extra spaces in this file. It's important to get the formatting exactly right!



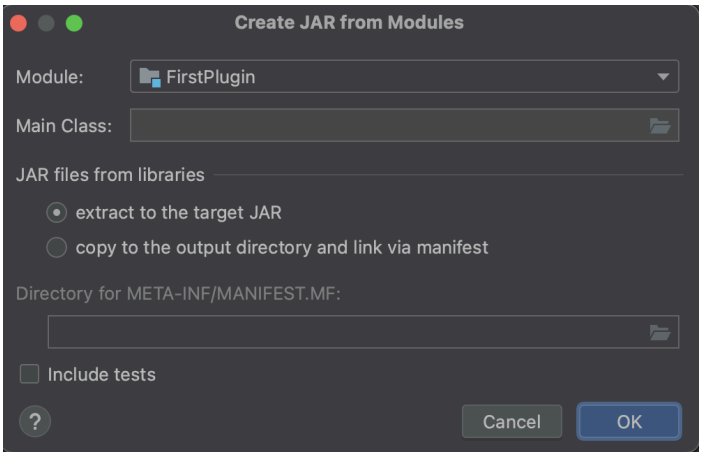


8 - Create the Build Artifact for your plugin

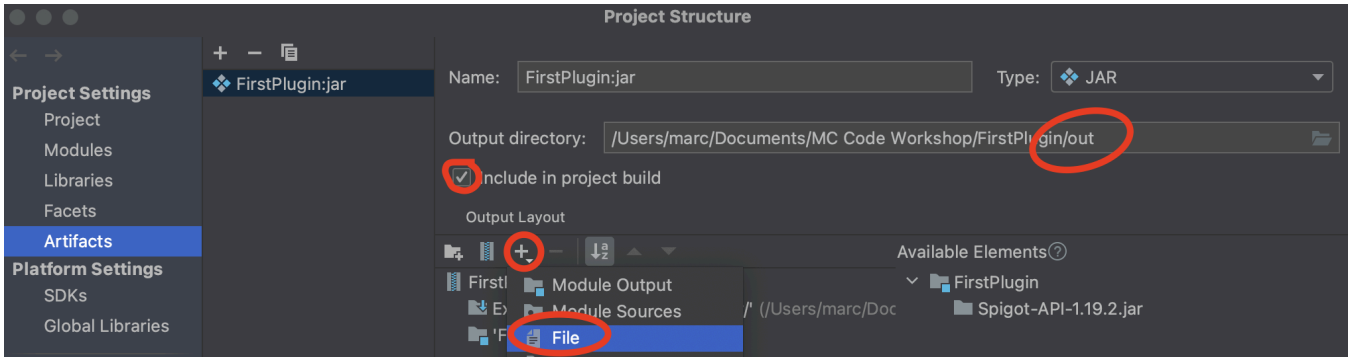
- The last step before you can make a FirstPlugin.jar file is to tell IntelliJ how to build the plugin.
- Open the **Project Structure** menu from earlier (File -> Project Structure). Under the Artifacts menu, click the plus sign, then JAR, then From modules with dependencies.



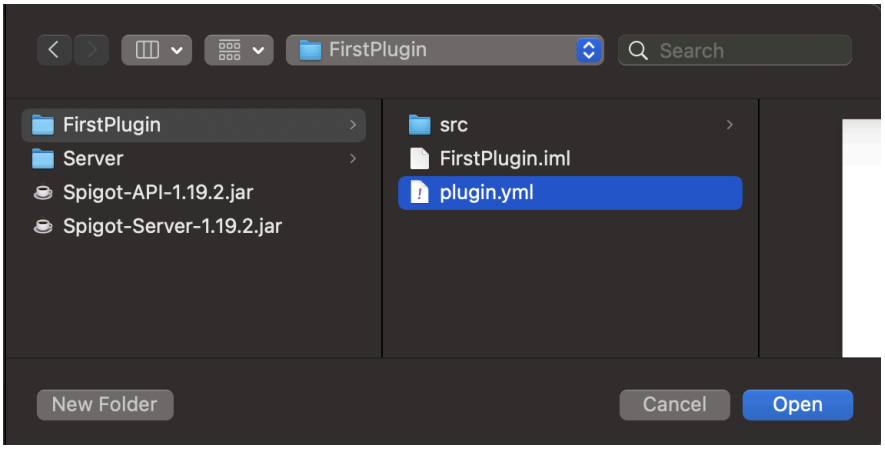
- In the menu that opens up, just click OK. No changes needed.



- Three more changes are needed here, though.
  - Make sure “Include in project build” is selected.
  - Remove “/artifacts/FirstPlugin_jar” from the end of the Output directory line. It should just end with “/out”.
  - Lastly, click the plus sign circled below and select File.

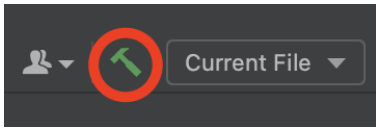


- Select the plugin.yml file you just created in the main folder of your project. IntelliJ may first show you somewhere completely different on your computer. Make sure you’re in the right folder.
- Once that’s added to the “Output Layout” menu, click Apply and OK.



9 - Build your plugin!

- Finally you can build your plugin to a JAR file! There are several ways you can build an IntelliJ project.
  - Go to “Build -> Build Project”.
  - Also, you can look for and click the icon shown to the right.



- When a project is building, a progress bar will show up in the lower right corner of the project window.

- If all goes well, the JAR file will be available under the “out” folder as you specified. Well done!