minigameslib.de

Spigot Eclipse-Plugin

Eclipse-Plugin and JUnit support

mepeisen
12.11.2016

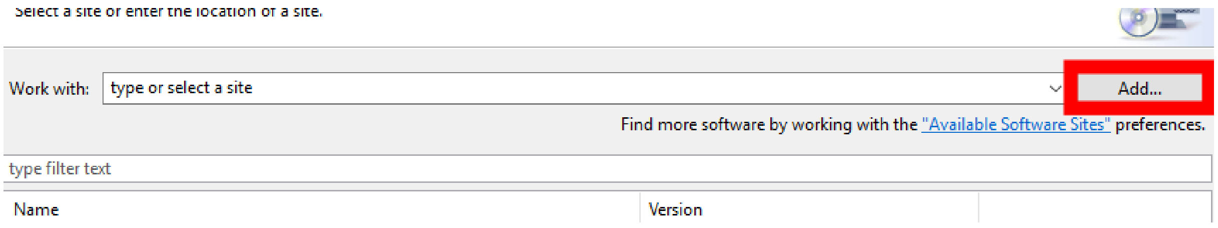
Table of contents

Eclipse setup

The spigot plugins were tested on Eclipse neon.

1) You can download eclipse neon for java EE development at http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon1a

2) After first start go to “help” > “Install new software”. Select the “Add” button on the right.

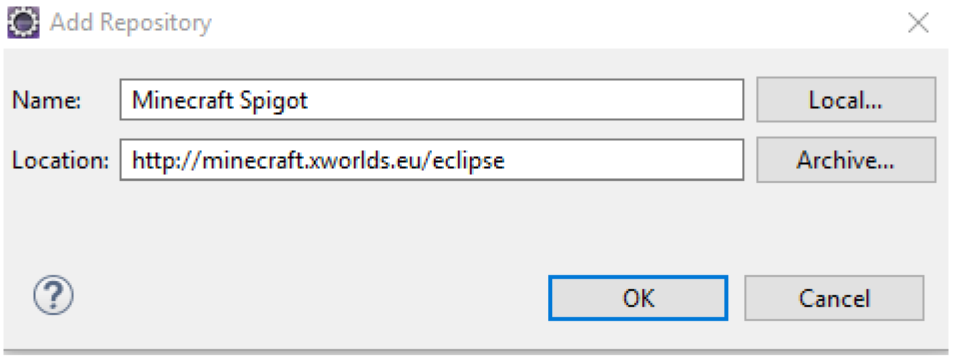

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

type filter text

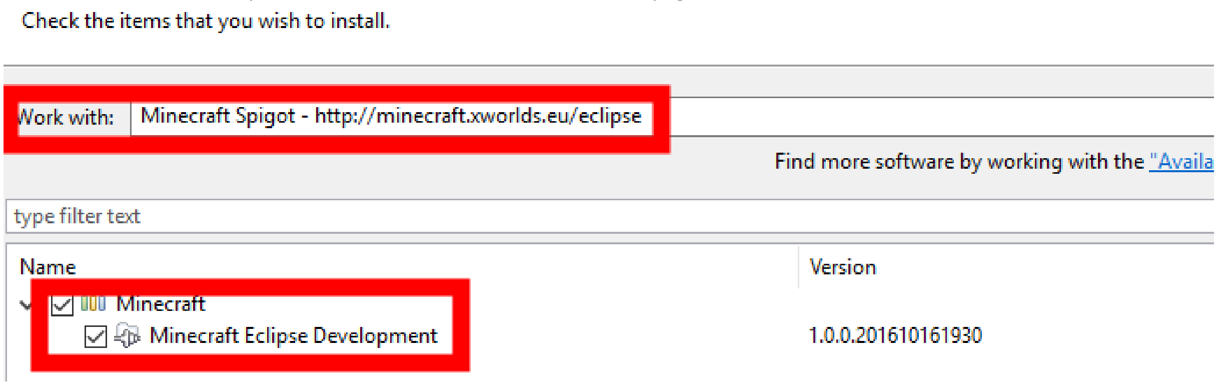
3) You can now enter a new update site with following values:
Name: Minecraft Spigot
Location: http://minecraft.xworlds.eu/eclipse


Add Repository

Name: Minecraft Spigot

Location: http://minecraft.xworlds.eu/eclipse

4) Select the site in drop down and select the minecraft spigot feature to install.


Check the items that you wish to install.

Work with: Minecraft Spigot - http://minecraft.xworlds.eu/eclipse

Find more software by working with the "Availa

type filter text

Name	Version
☑ Minecraft	
☑ Minecraft Eclipse Development	1.0.0.201610161930

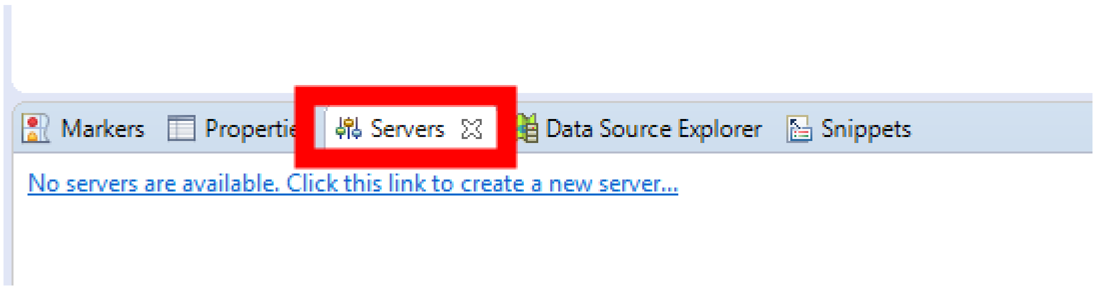
5) Follow the wizard steps and let eclipse install the plugins.

Setup spigot servers

Eclipse webtools divides servers and runtimes. A runtime is a location of your spigot server on your local hdd. A server holds all configurations, plugins and everything you need to start spigot.

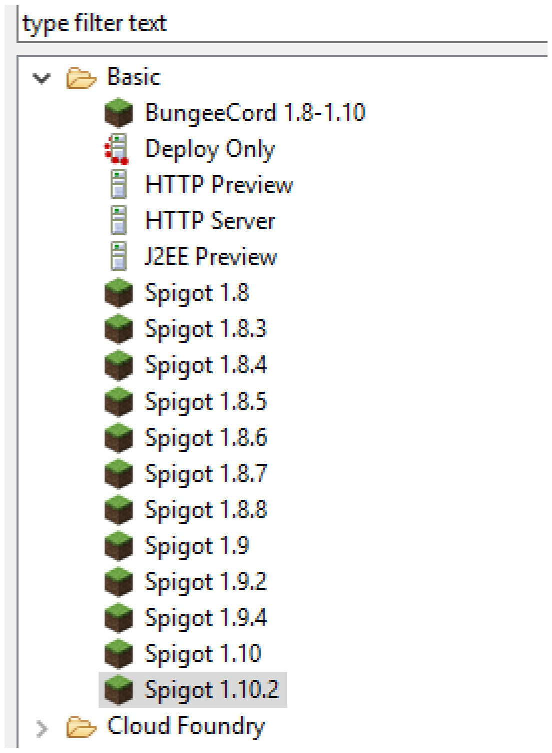
If you are a developer of two plugins you can use the same runtime within two servers. It is one server for each plugin. Or you can use only one server for both.

1) To create a server, go to the server view on the bottom of the java ee perspective:



No servers are available. Click this link to create a new server...

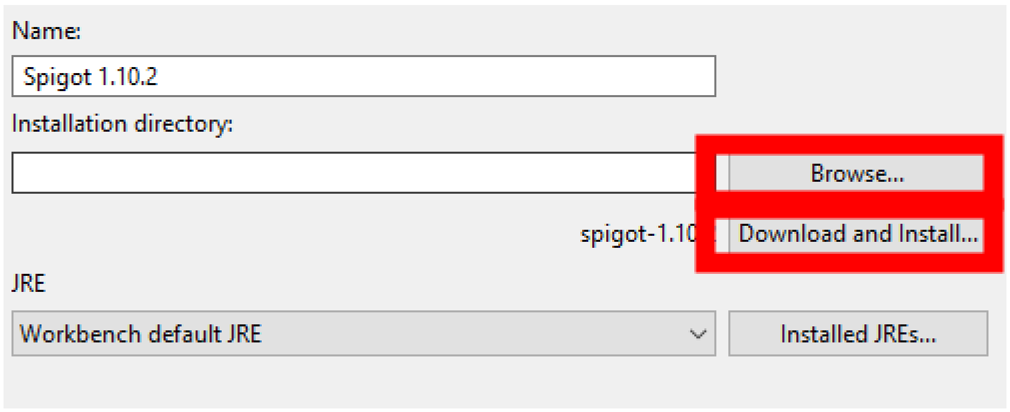
2) Click on the hyperlink. In the “new Server” wizard select the spigot server under category “Basic”.



type filter text

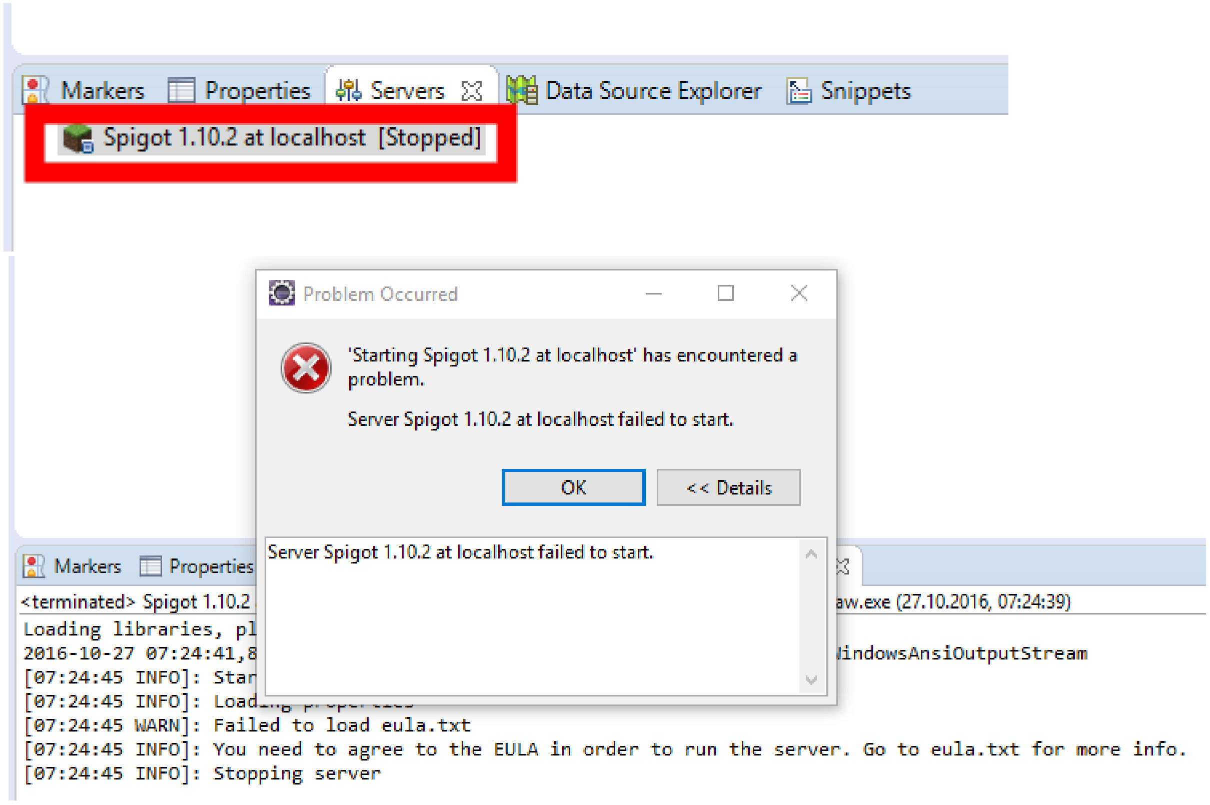
- Basic
  - BungeeCord 1.8-1.10
  - Deploy Only
  - HTTP Preview
  - HTTP Server
  - J2EE Preview
  - Spigot 1.8
  - Spigot 1.8.3
  - Spigot 1.8.4
  - Spigot 1.8.5
  - Spigot 1.8.6
  - Spigot 1.8.7
  - Spigot 1.8.8
  - Spigot 1.9
  - Spigot 1.9.2
  - Spigot 1.9.4
  - Spigot 1.10
  - Spigot 1.10.2
- Cloud Foundry

3) You will be forces to create a new runtime. Select either your existing installation or download it from our repository.



4) On the next wizard page it will ask you which resources to add. We do not yet have any spigot plugin in our workspace. Simply ignore and click on “finish”.

5) Now you have a working spigot server. Right click or use the buttons on the right to start/debug it.



'Starting Spigot 1.10.2 at localhost' has encountered a problem.

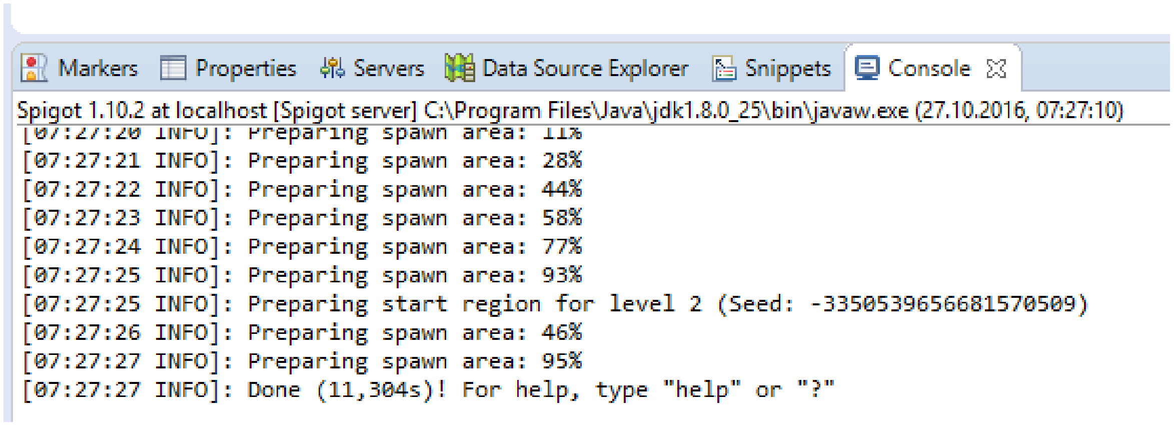
Server Spigot 1.10.2 at localhost failed to start.

Server Spigot 1.10.2 at localhost failed to start.

```
<terminated> Spigot 1.10.2                                  aw.exe (27.10.2016, 07:24:39)
Loading libraries, pl
2016-10-27 07:24:41,8                                      WindowsAnsiOutputStream
[07:24:45 INFO]: Star
[07:24:45 INFO]: Load
[07:24:45 WARN]: Failed to load eula.txt
[07:24:45 INFO]: You need to agree to the EULA in order to run the server. Go to eula.txt for more info.
[07:24:45 INFO]: Stopping server
```

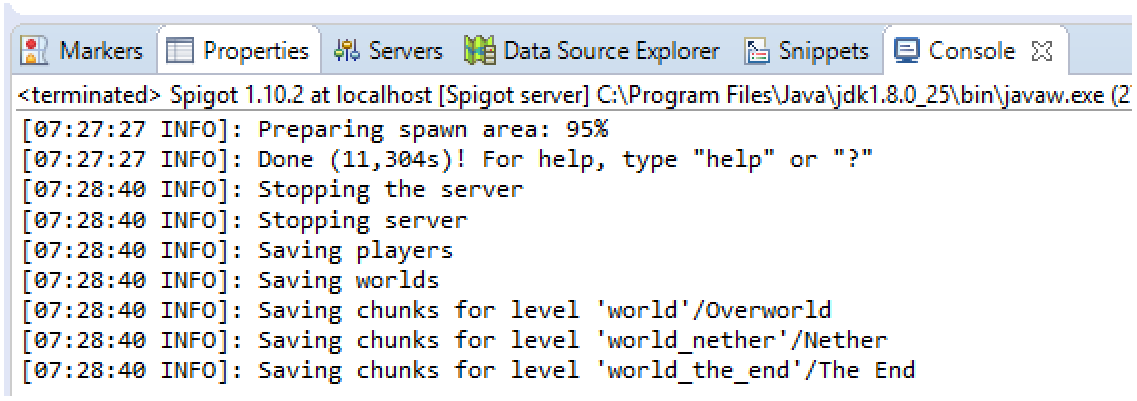
6) As you see it will fail. However as an experienced minecraft/ spigot user you will know what to do. Have a look at your workspace and open the “Servers” projects. After doing a refresh

(F5) you will see the eula.txt. Edit it, accept the eula and restart your server again.



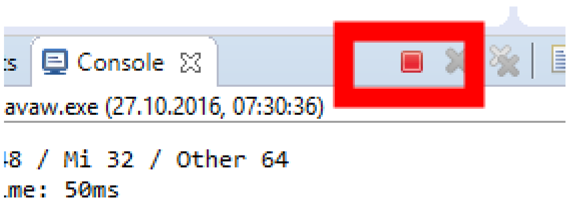
```
Spigot 1.10.2 at localhost [Spigot server] C:\Program Files\Java\jdk1.8.0_25\bin\javaw.exe (27.10.2016, 07:27:10)
[07:27:20 INFO]: Preparing spawn area: 11%
[07:27:21 INFO]: Preparing spawn area: 28%
[07:27:22 INFO]: Preparing spawn area: 44%
[07:27:23 INFO]: Preparing spawn area: 58%
[07:27:24 INFO]: Preparing spawn area: 77%
[07:27:25 INFO]: Preparing spawn area: 93%
[07:27:25 INFO]: Preparing start region for level 2 (Seed: -3350539656681570509)
[07:27:26 INFO]: Preparing spawn area: 46%
[07:27:27 INFO]: Preparing spawn area: 95%
[07:27:27 INFO]: Done (11,304s)! For help, type "help" or "?"
```

7) To shut down the server simply right click in the servers view or type “stop” in the console. You may use your minecraft to connect to the server and type “/stop” in the chat as well.

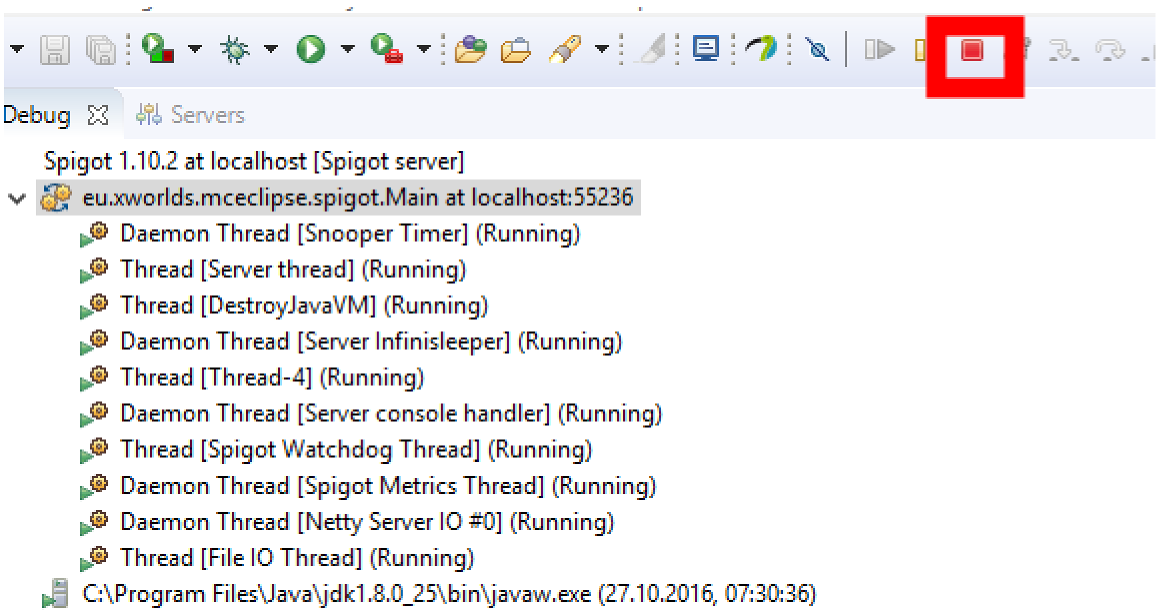


```
<terminated> Spigot 1.10.2 at localhost [Spigot server] C:\Program Files\Java\jdk1.8.0_25\bin\javaw.exe (2
[07:27:27 INFO]: Preparing spawn area: 95%
[07:27:27 INFO]: Done (11,304s)! For help, type "help" or "?"
[07:28:40 INFO]: Stopping the server
[07:28:40 INFO]: Stopping server
[07:28:40 INFO]: Saving players
[07:28:40 INFO]: Saving worlds
[07:28:40 INFO]: Saving chunks for level 'world'/Overworld
[07:28:40 INFO]: Saving chunks for level 'world_nether'/Nether
[07:28:40 INFO]: Saving chunks for level 'world_the_end'/The End
```

NOTICE: In the console view there is a red terminate button. Do not use it. It will shut down the java VM but does not save world data etc. You may use it if the spigot server crashed and does not respond. You should always prefer stopping the server by “stop” command or via “Servers” view. The second non-preferred terminate button is inside debug view.



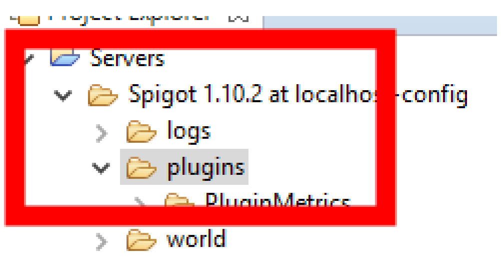
```
avaw.exe (27.10.2016, 07:30:36)
!8 / Mi 32 / Other 64
.me: 50ms
```


Debug Servers

Spigot 1.10.2 at localhost [Spigot server]
eu.xworlds.mceclipse.spigot.Main at localhost:55236
Daemon Thread [Snooper Timer] (Running)
Thread [Server thread] (Running)
Thread [DestroyJavaVM] (Running)
Daemon Thread [Server Infinisleeper] (Running)
Thread [Thread-4] (Running)
Daemon Thread [Server console handler] (Running)
Thread [Spigot Watchdog Thread] (Running)
Daemon Thread [Spigot Metrics Thread] (Running)
Daemon Thread [Netty Server IO #0] (Running)
Thread [File IO Thread] (Running)
C:\Program Files\Java\jdk1.8.0_25\bin\javaw.exe (27.10.2016, 07:30:36)

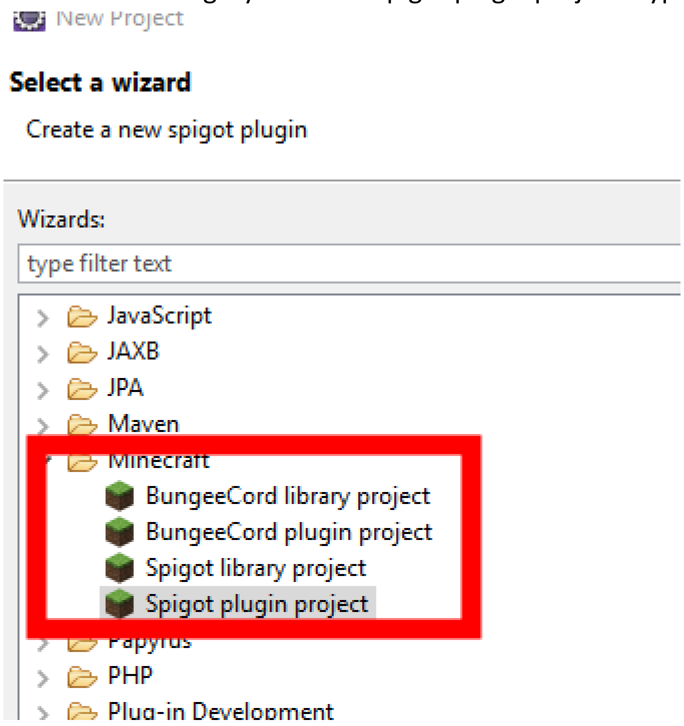
Installing third party plugins

It is fairly simple. Stop the server. Download the plugin jar files. Copy them. Open the Servers project and put them into the plugins folder.



First plugin project

1) Right click in the "project explorer" view and select "New" > "Project". You will find a "Minecraft" category and the "Spigot plugin project" type.


New Project

Select a wizard

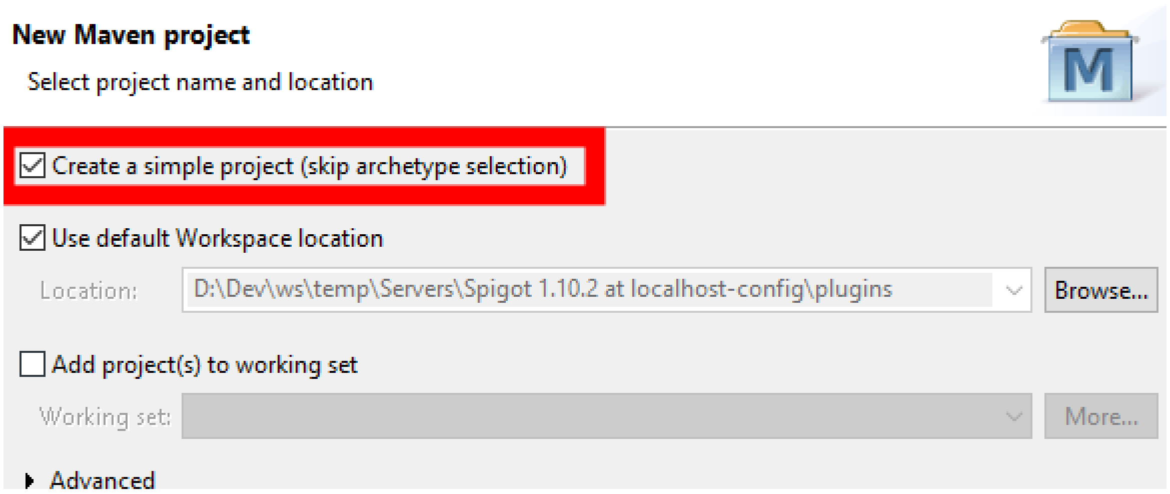
Create a new spigot plugin

Wizards:

type filter text

- JavaScript
- JAXB
- JPA
- Maven
- Minecraft
  - BungeeCord library project
  - BungeeCord plugin project
  - Spigot library project
  - Spigot plugin project
- Papyrus
- PHP
- Plug-in Development

2) Select the "Create a simple project" checkbox.


New Maven project

Select project name and location

☑ Create a simple project (skip archetype selection)

☑ Use default Workspace location

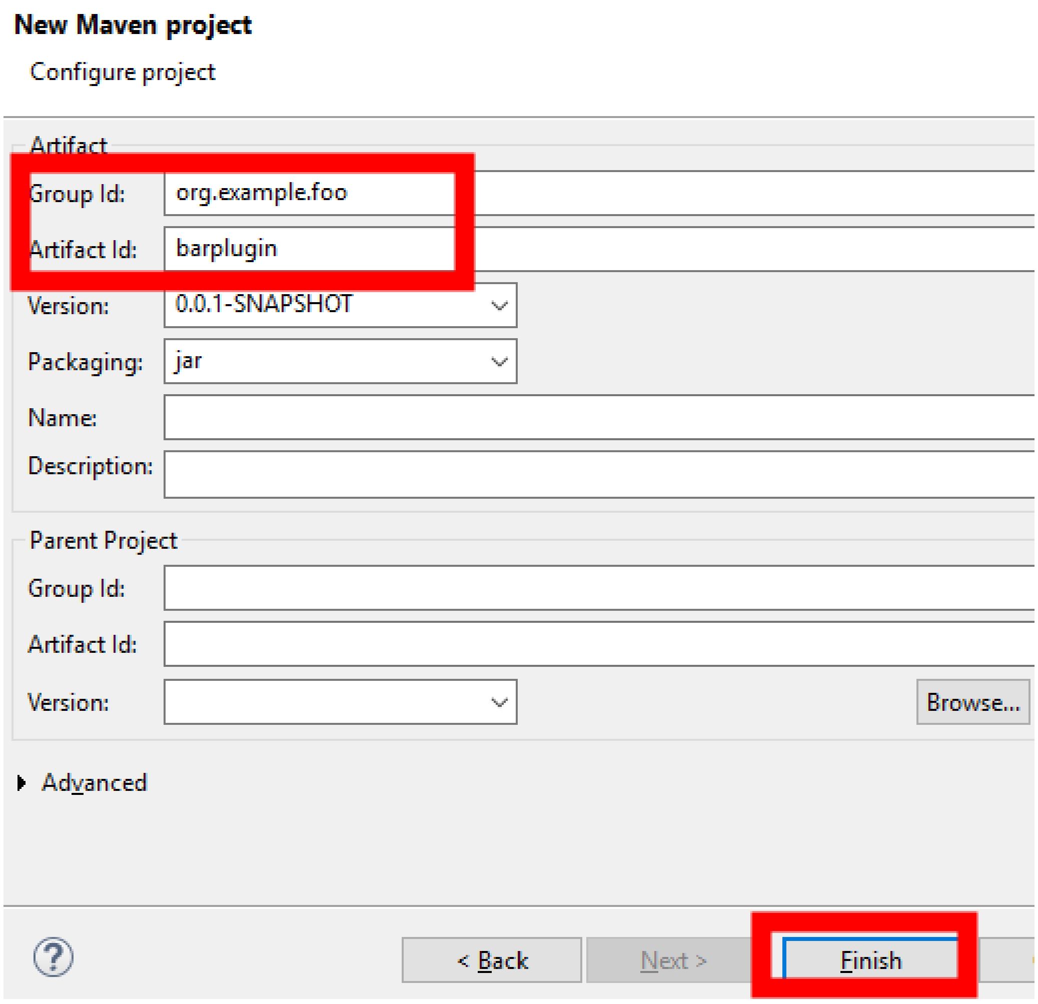
Location: D:\Dev\ws\temp\Servers\Spigot 1.10.2 at localhost-config\plugins [Browse...]

☐ Add project(s) to working set

Working set: [More...]

Advanced

3) Type a maven group id (typically a reversed internet domain you own). Type a maven artifact id and select “Finish”.



New Maven project

Configure project

Artifact

Group Id: org.example.foo

Artifact Id: barplugin

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

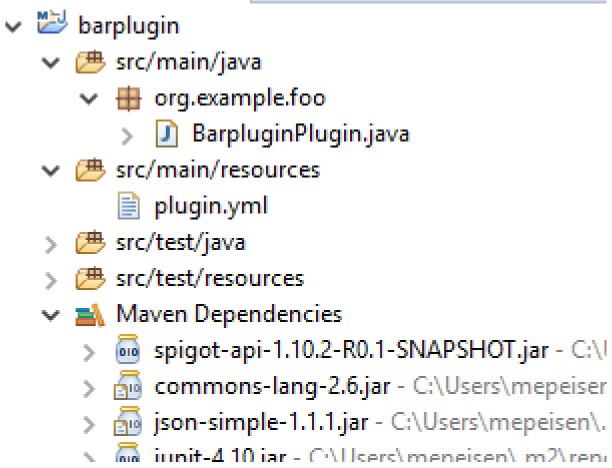
Parent Project

Group Id:

Artifact Id:

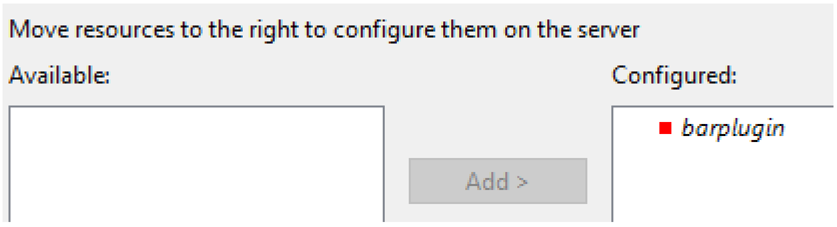
Version:

4) You now have a new spigot plugin within your workspace.



```
barplugin
  src/main/java
    org.example.foo
      BarpluginPlugin.java
  src/main/resources
    plugin.yml
  src/test/java
  src/test/resources
  Maven Dependencies
    spigot-api-1.10.2-R0.1-SNAPSHOT.jar - C:\
    commons-lang-2.6.jar - C:\Users\mepeiser
    json-simple-1.1.1.jar - C:\Users\mepeisen\.
    junit-4.10.jar - C:\Users\mepeisen\.m2\rep
```

5) Go back to your server and right click on it. Select the “Add and remove” command. You will now see the barplugin being available for your spigot server. Move it to the right.

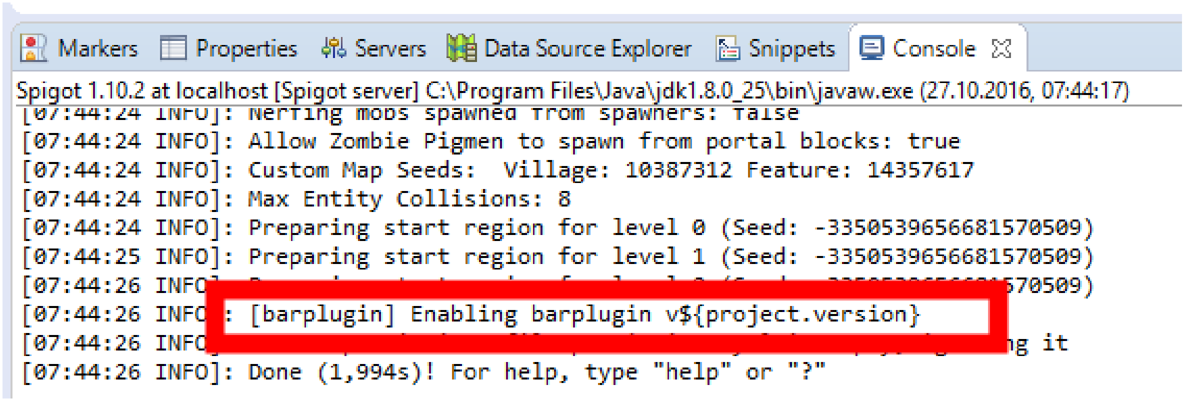

Move resources to the right to configure them on the server

Available:

Configured:

barplugin

6) Restart the server and watch the console.

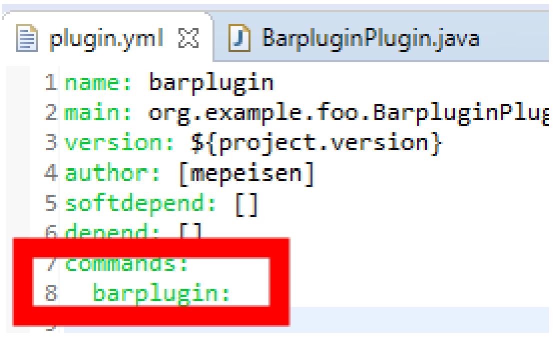


Spigot 1.10.2 at localhost [Spigot server] C:\Program Files\Java\jdk1.8.0_25\bin\javaw.exe (27.10.2016, 07:44:17)

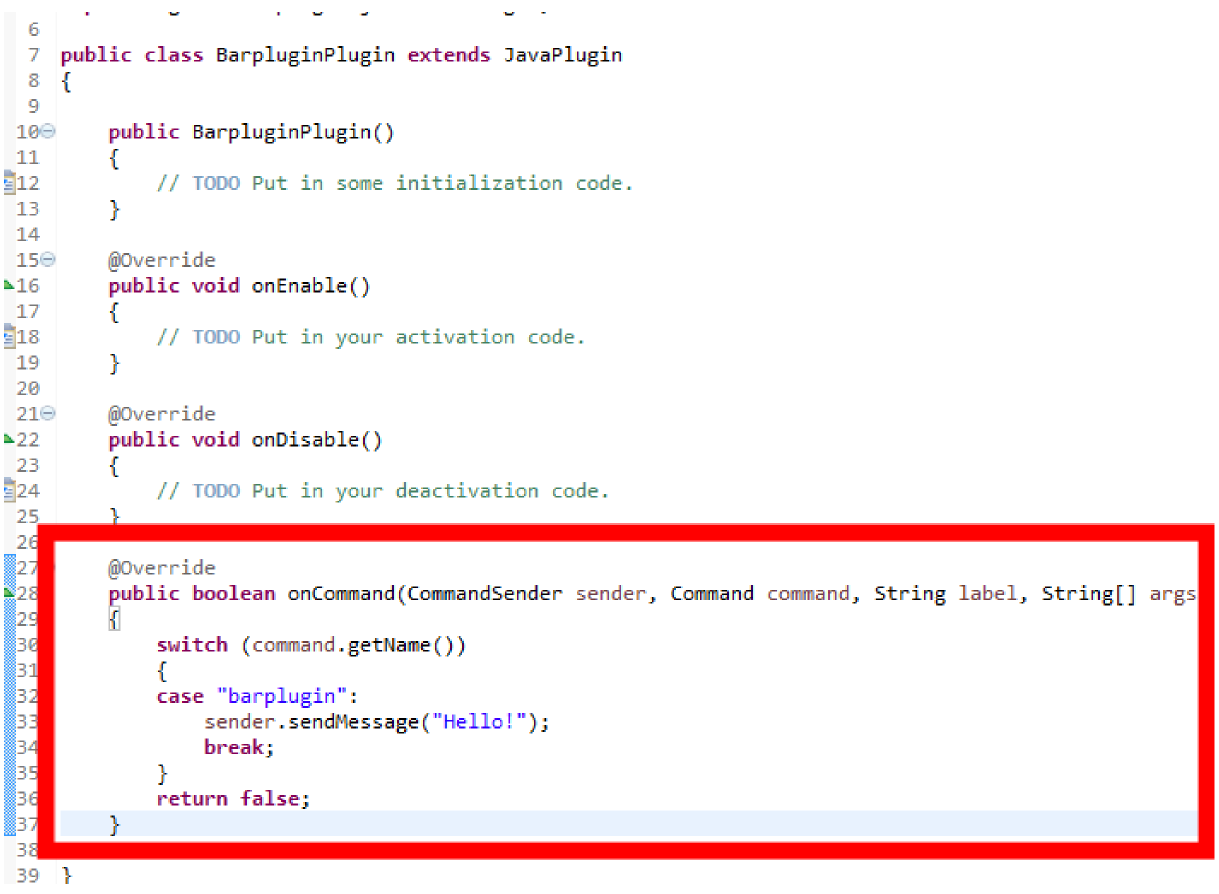
```
[07:44:24 INFO]: Nerfing mobs spawned from spawners: false
[07:44:24 INFO]: Allow Zombie Pigmen to spawn from portal blocks: true
[07:44:24 INFO]: Custom Map Seeds:  Village: 10387312 Feature: 14357617
[07:44:24 INFO]: Max Entity Collisions: 8
[07:44:24 INFO]: Preparing start region for level 0 (Seed: -3350539656681570509)
[07:44:25 INFO]: Preparing start region for level 1 (Seed: -3350539656681570509)
[07:44:26 INFO                                                        70509)
[07:44:26 INFO]: [barplugin] Enabling barplugin v${project.version}
[07:44:26 INFO                                                       ng it
[07:44:26 INFO]: Done (1,994s)! For help, type "help" or "?"
```

Hot code replacement.

Eclipse supports a feature called hot code replacement. That’s useful during development. It allows you to change code without restarting the whole server. We will create a small useful example. Add a command by editing the plugin.yml and the BarpluginPlugin class.

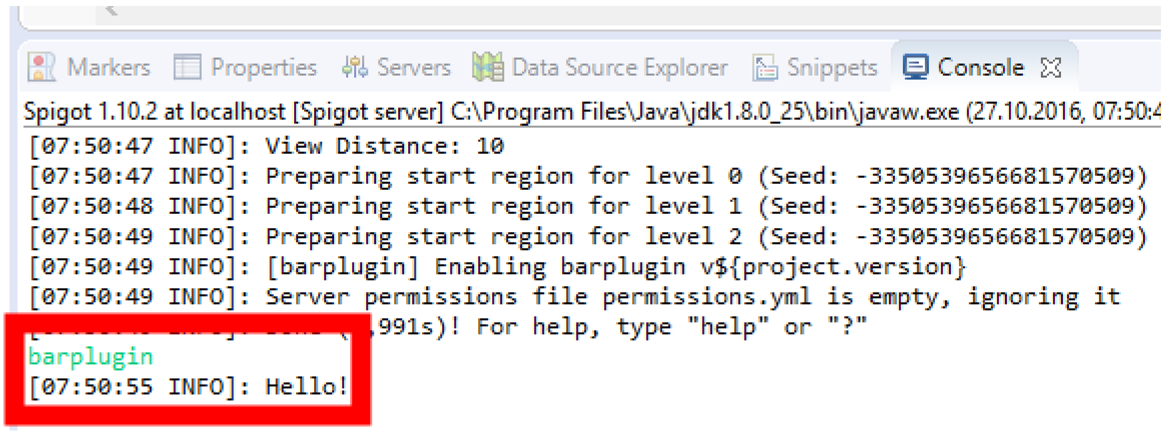


```
name: barplugin
main: org.example.foo.BarpluginPlug
version: ${project.version}
author: [mepeisen]
softdepend: []
depend: []
commands:
  barplugin:
```



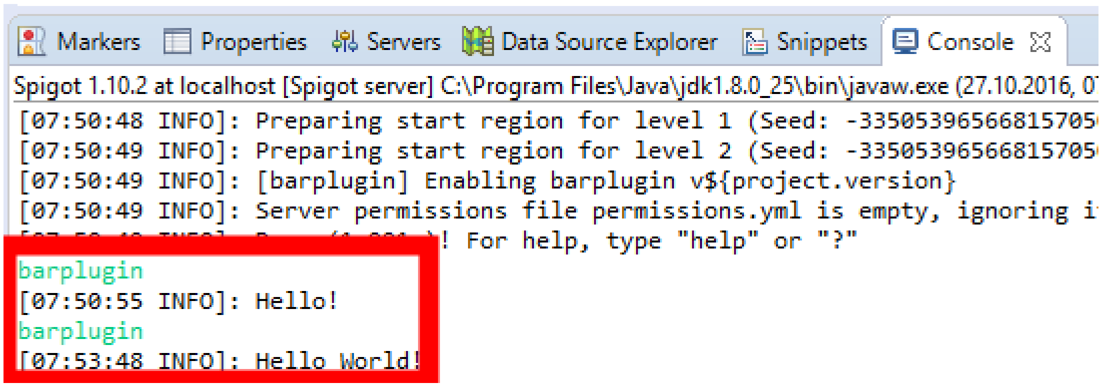
```java

public class BarpluginPlugin extends JavaPlugin
{

    public BarpluginPlugin()
    {
        // TODO Put in some initialization code.
    }

    @Override
    public void onEnable()
    {
        // TODO Put in your activation code.
    }

    @Override
    public void onDisable()
    {
        // TODO Put in your deactivation code.
    }

    @Override
    public boolean onCommand(CommandSender sender, Command command, String label, String[] args
    {
        switch (command.getName())
        {
        case "barplugin":
            sender.sendMessage("Hello!");
            break;
        }
        return false;
    }

}
```

Now start the server and check your code.



```
Spigot 1.10.2 at localhost [Spigot server] C:\Program Files\Java\jdk1.8.0_25\bin\javaw.exe (27.10.2016, 07:50:4
[07:50:47 INFO]: View Distance: 10
[07:50:47 INFO]: Preparing start region for level 0 (Seed: -3350539656681570509)
[07:50:48 INFO]: Preparing start region for level 1 (Seed: -3350539656681570509)
[07:50:49 INFO]: Preparing start region for level 2 (Seed: -3350539656681570509)
[07:50:49 INFO]: [barplugin] Enabling barplugin v${project.version}
[07:50:49 INFO]: Server permissions file permissions.yml is empty, ignoring it
                                  991s)! For help, type "help" or "?"
barplugin
[07:50:55 INFO]: Hello!
```

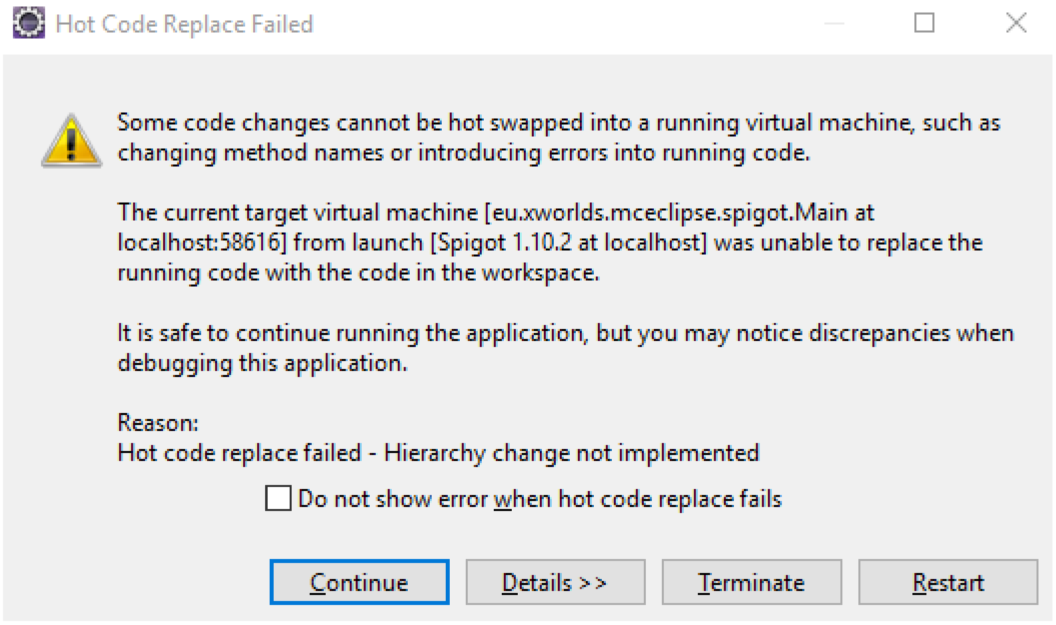
It is working. But let us now use the hot code replacement. WIHTOUT restarting the server change the message in your code to “Hello World!”. Save the java file and check the command again.



```
Spigot 1.10.2 at localhost [Spigot server] C:\Program Files\Java\jdk1.8.0_25\bin\javaw.exe (27.10.2016, 0
[07:50:48 INFO]: Preparing start region for level 1 (Seed: -33505396566815705
[07:50:49 INFO]: Preparing start region for level 2 (Seed: -33505396566815705
[07:50:49 INFO]: [barplugin] Enabling barplugin v${project.version}
[07:50:49 INFO]: Server permissions file permissions.yml is empty, ignoring i
                                          ! For help, type "help" or "?"
barplugin
[07:50:55 INFO]: Hello!
barplugin
[07:53:48 INFO]: Hello World!
```

The message changed. Without building jar files, without redeploying to server and without restarting the server.

The hot code replacement is limited. Changing class structures often result in warnings to restart the server. For example let your plugin implement the bukkit listener interface. You will get the following warning.



Some code changes cannot be hot swapped into a running virtual machine, such as changing method names or introducing errors into running code.

The current target virtual machine [eu.xworlds.mceclipse.spigot.Main at localhost:58616] from launch [Spigot 1.10.2 at localhost] was unable to replace the running code with the code in the workspace.

It is safe to continue running the application, but you may notice discrepancies when debugging this application.

Reason:
Hot code replace failed - Hierarchy change not implemented

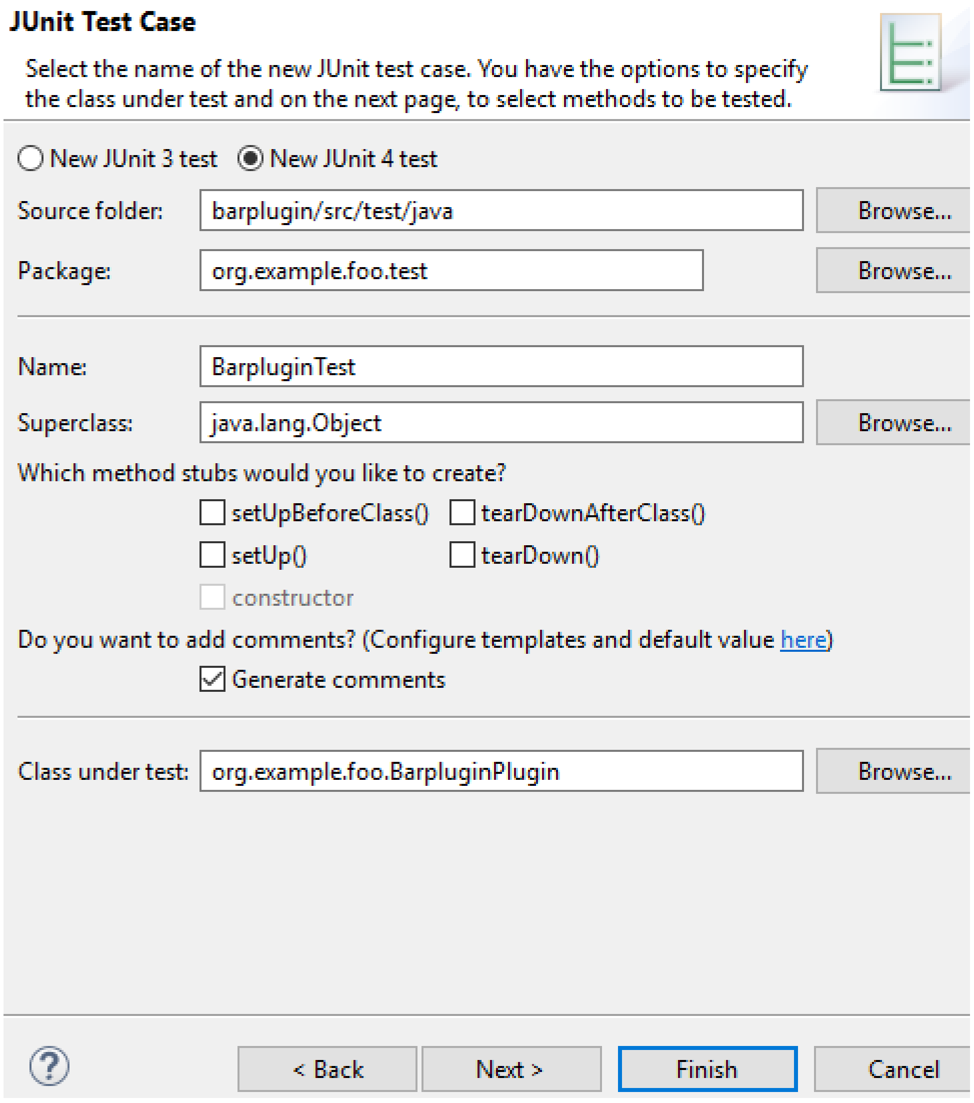
Your spigot server is now out of sync. Restart it to get test the new code.

Junit support

1) Edit your pom.xml and add the following elements:

```
        <dependency>
            <groupId>de.minigameslib.mclib</groupId>
            <artifactId>spigot-testsupport</artifactId>
            <version>0.0.1-SNAPSHOT</version>
            <scope>test</scope>
        </dependency>

  </dependencies>
  <repositories>
        <repository>
            <id>mce-repo</id>
            <url>http://nexus.xworlds.eu/nexus/content/groups/mce</url>
        </repository>
```

2) Right click on "src/test/java" and create a JUnit class for testing ("New " > "Other" > "Java" > "Junit" > "Junit Test Case").



JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

○ New JUnit 3 test [✓ ● New JUnit 4 test]

Source folder:	barplugin/src/test/java
Package:	org.example.foo.test
Name:	BarpluginTest
Superclass:	java.lang.Object

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☐ setUp() ☐ tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value here)

☑ Generate comments

Class under test: org.example.foo.BarpluginPlugin

3) Add the annotations “RunsWith” and “SpigotTest” at class level.

```
 */
@RunWith(SpigotJunit4Runner.class)
@SpigotTest(versions = {SpigotVersion.V1_10_2})
public class BarpluginTest {
```

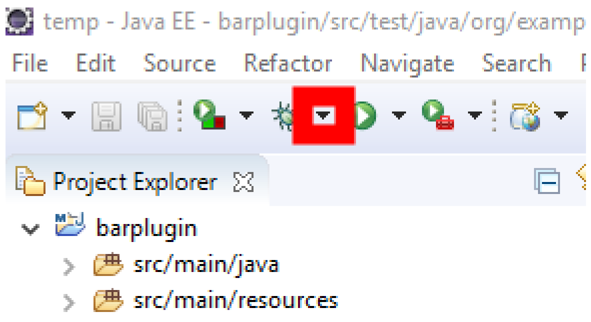
4) To access our spigot server during junit tests we need to add a variable annotated with “SpigotInject”.

```
@SpigotInject
private SpigotServer server;
```

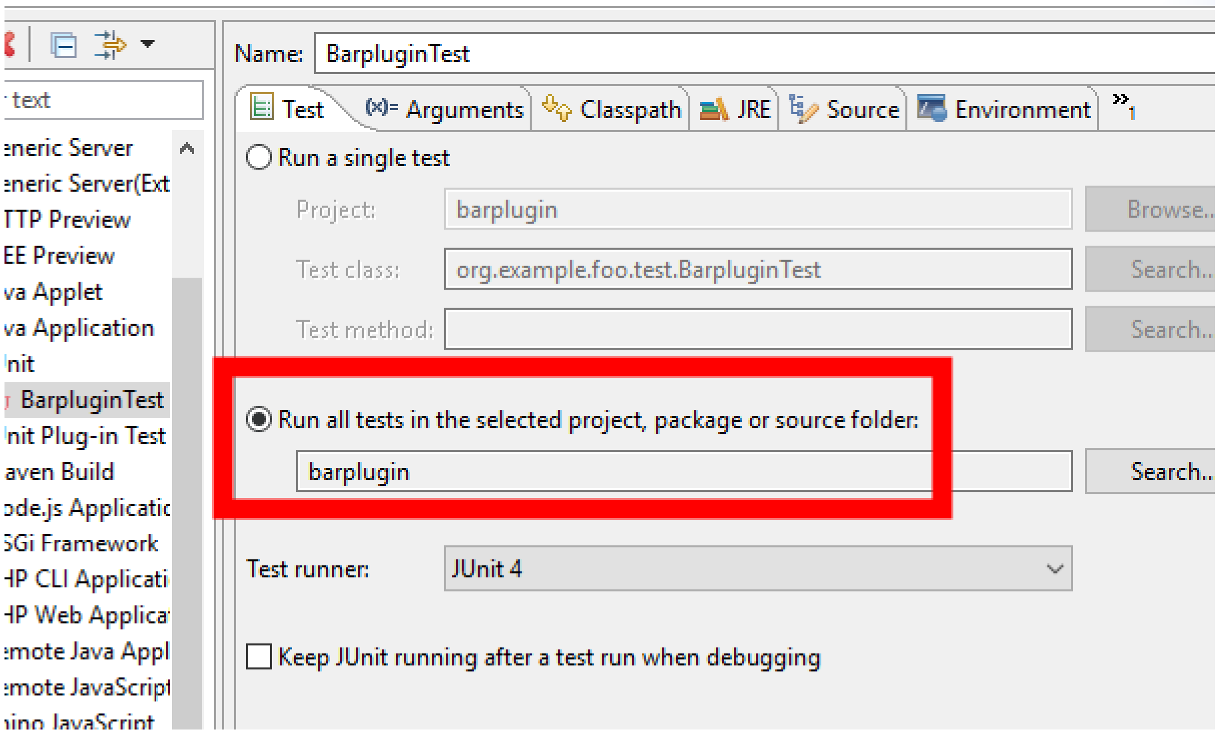
5) We are now ready to perform our test. As we already have a command we will simply test it:

```
@Test
public void testBarpluginCommand() throws IOException
{
    this.server.clearConsole();
    this.server.sendCommand("barplugin");
    assertTrue(this.server.waitForConsole(".*Hello.*", 5000));
}
```

6) We now create a new debug profile via toolbar. Click the small arrow right from the bug and select “debug configurations…”.



7) Right click on "Junit" and select "New". Change the properties to run all tests from your project.



Name: BarpluginTest

( ) Run a single test

Project: barplugin

Test class: org.example.foo.test.BarpluginTest

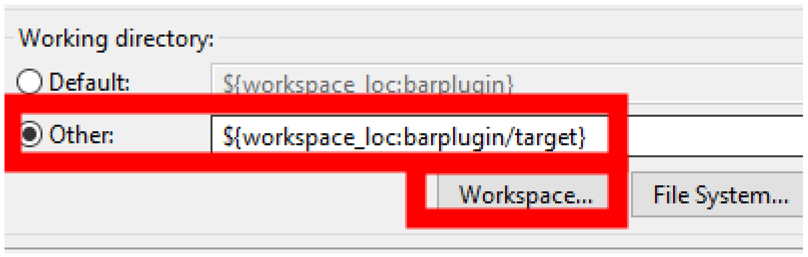
Test method:

(●) Run all tests in the selected project, package or source folder: barplugin

Test runner: JUnit 4

☐ Keep JUnit running after a test run when debugging

8) Change the arguments to start in directory "target".

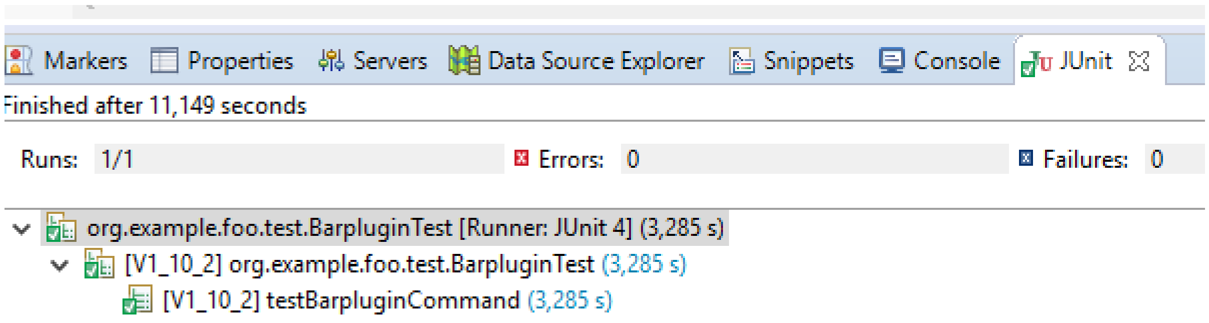


Working directory:

( ) Default: ${workspace_loc:barplugin}

(●) Other: ${workspace_loc:barplugin/target}

9) Start the test by selecting the debug button.



```
Finished after 11,149 seconds

Runs: 1/1        Errors: 0        Failures: 0

org.example.foo.test.BarpluginTest [Runner: JUnit 4] (3,285 s)
  [V1_10_2] org.example.foo.test.BarpluginTest (3,285 s)
    [V1_10_2] testBarpluginCommand (3,285 s)
```

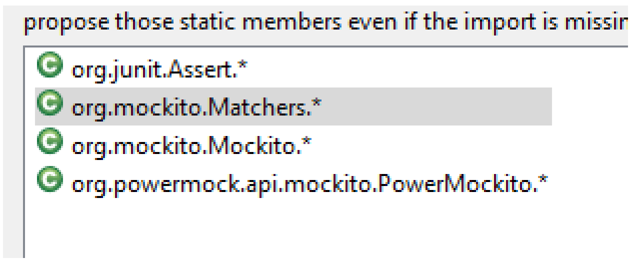
Hint: We do not really need our own debug configuration for the whole project and with folder "target".

You can simply run your tests by right clicking the class and select "Run as Junit test".

However eclipse always chooses to use the projects root directory. You may be messed up with log files, configuration files etc.

Typically in scm (SVN or GIT) you will never commit the target directory. Choosing the target directory as test working directory will automatically hide all the test files from scm.

Hint: We use a static import "assertTrue" in our sample code. Eclipse is able to automatically import statics but the feature is somehow hard to find. Go the "Window" > "Preferences" > "Java" > "Editor" > "Content Assist" > "Favorites". Add the following by selecting the "New Type" button:

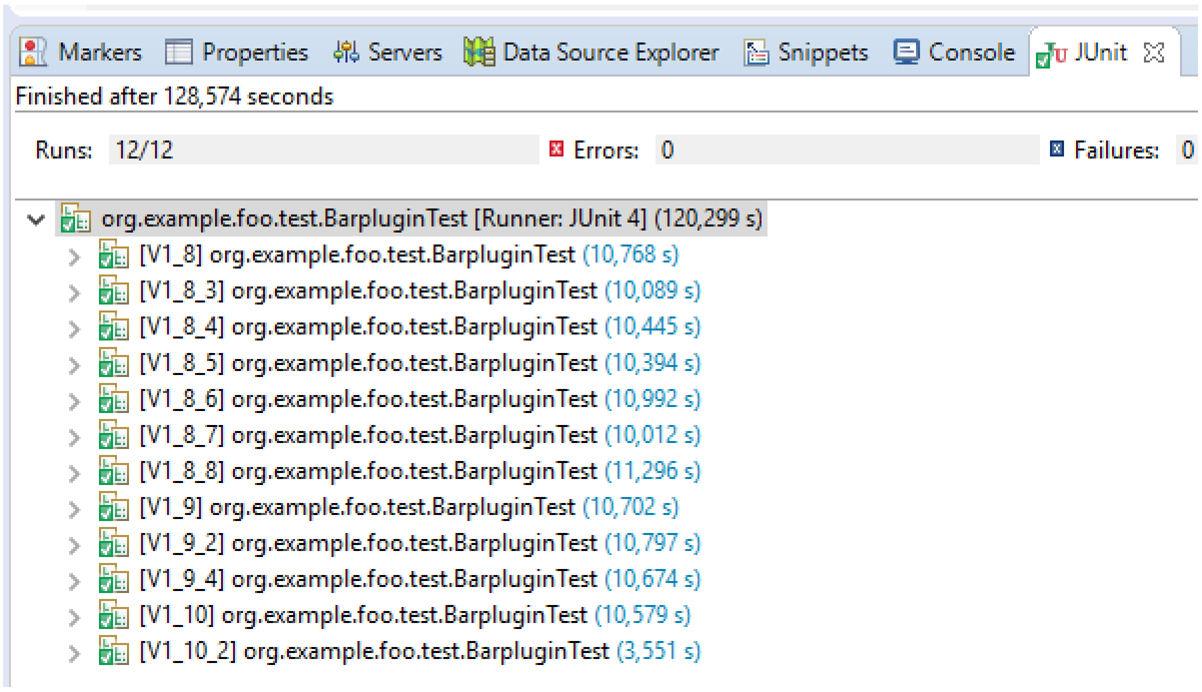


propose those static members even if the import is missir

- org.junit.Assert.*
- org.mockito.Matchers.*
- org.mockito.Mockito.*
- org.powermock.api.mockito.PowerMockito.*

Now you can type "assertTrue" in the java code editor and press "Ctrl+Space" for content assist and creating a static import.

Hint: As you see the SpigotTest annotation controls the servers you want to test. You can add more servers or even simply set "all" to true. See what happens if we test all the versions:



```
Finished after 128,574 seconds

Runs: 12/12          Errors: 0          Failures: 0

org.example.foo.test.BarpluginTest [Runner: JUnit 4] (120,299 s)
  [V1_8] org.example.foo.test.BarpluginTest (10,768 s)
  [V1_8_3] org.example.foo.test.BarpluginTest (10,089 s)
  [V1_8_4] org.example.foo.test.BarpluginTest (10,445 s)
  [V1_8_5] org.example.foo.test.BarpluginTest (10,394 s)
  [V1_8_6] org.example.foo.test.BarpluginTest (10,992 s)
  [V1_8_7] org.example.foo.test.BarpluginTest (10,012 s)
  [V1_8_8] org.example.foo.test.BarpluginTest (11,296 s)
  [V1_9] org.example.foo.test.BarpluginTest (10,702 s)
  [V1_9_2] org.example.foo.test.BarpluginTest (10,797 s)
  [V1_9_4] org.example.foo.test.BarpluginTest (10,674 s)
  [V1_10] org.example.foo.test.BarpluginTest (10,579 s)
  [V1_10_2] org.example.foo.test.BarpluginTest (3,551 s)
```

Dependencies

First of all some theory of how maven manages dependencies. The most important thing is the dependency scope.

You can read details on the following web page: https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html

We are focusing on three scopes: compile, provided, runtime.

The compile scope is the default. We use it for all non-minecraft jar files. For example you can add apaches commons-mail. Adding it with compile scope (default) will cause the eclipse plugin to add it to the plugin classpath.

The provided classpath is more special. We use it for all plugins and plugin apis that are minecraft related. Provided classpath causes maven to add it during compile time (you can use it) but not to the classpath itself. We use this scope for spigot-api itself too. Evenrything that is not part or your plugin will at least use "provided" or "runtime".

The runtime classpath is only a hint for maven that this dependency must be used while running the server.

How does eclipse minecraft plugin use the scopes?

It is really simple:

1) All external jar files are using no scope/ "compile" scope.
2) All Plugin APIs or Plugin Jars you need during compile will use "provided" scope.
3) All Plugin Jars implementing a plugin API will use "runtime" scope.

Some simple example:

```
<dependency>
   <groupId>org.apache.commons</groupId>
   <artifactId>commons-email</artifactId>
   <version>1.4</version>
</dependency>
<dependency>
   <groupId>de.minigameslib.mclib</groupId>
   <artifactId>mclib-api</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>de.minigameslib.mclib</groupId>
   <artifactId>mclib</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <scope>runtime</scope>
</dependency>
```

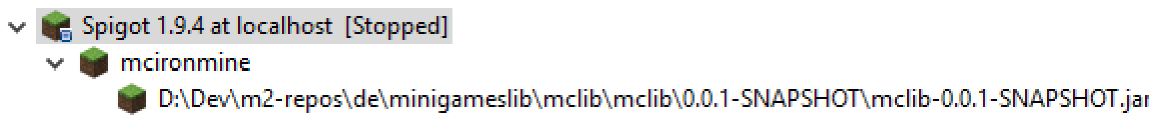
What do we do here?

First we add the apache mailing library. It is a compile library and we like to embed it into our plugin.

As second dependency we add minigames mclib. It uses provided scope. We will see it during compile time so we can develop against it. But we do not want it to be part of our plugin. Instead the api will be shipped with mclib jar and the user of our plugin will have to install both, our plugin and mclib.

At third position we give maven and the eclipse plugin some hint. We require the presence of "mclib" during runtme. We do not compile against it but we need it to work.

Let us view the result:



```
Spigot 1.9.4 at localhost [Stopped]
  mcironmine
    D:\Dev\m2-repos\de\minigameslib\mclib\mclib\0.0.1-SNAPSHOT\mclib-0.0.1-SNAPSHOT.jar
```

The eclipse plugin will automatically detect the dependencies. It will see that you use some other plugin in the dependencies and it will automatically install both plugins in your server.

Some word about maven-shade plugin

For spigot it is recommended to distribute a single jar file to your users containing everything that is relevant.

Given our previous example we want maven to build a jar file containing everything but mclib.

Add the following section to your pom.xml:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>2.4.3</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals><goal>shade</goal></goals>
          <configuration>
            <artifactSet>
              <excludes>
                <exclude>de.minigameslib.mclib:*</exclude>
              </excludes>
            </artifactSet>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

If you now invoke the maven build it will create a single jar file containing all the needed dependencies. But it excludes the mclib plugin.