Team: pay4grade
Duong, Harry
Nguyen, Thuc
Zhao, Deon

Managing Plugins: How to Navigate and Understand PluginManager.java

We found that the PluginManager class handles the loading and addition of plugins for the current session. This is a core feature of the system because the entire client displays plugins that the user can use to facilitate their gameplay, and this manager handles all of these plugins.

Change to the top-level directory where you downloaded Runeline. If you are in IntelliJ, the Project tab on the left will already have the hierarchy ready for you. Open the file using the path listed below.

Note that the Runelite client separates its plugins into separate and individual packages.

PluginManager - Adding a new plugin

Path: runelite-client/src/main/java/net/runelite/client/plugins/PluginManager.java

Within loadExternalPlugins() on line 98, loadPlugins() is called:

```
91      public void loadExternalPlugins() throws PluginInstantiationException
92      {
93          refreshPlugins();
94
95          if (builtinExternals != null)
96          {
97              // builtin external's don't actually have a manifest or a separate classloader...
98              pluginManager.loadPlugins(Lists.newArrayList(builtinExternals),  onPluginLoaded: null);
99          }
100     }
```

A second file to look at is ExternalPluginManager (runelite-client/src/main/java/net/runelite/client/externalplugins/ExternalPluginManager.java), which contains a PluginManager as a state variable, and uses this variable to load plugins in loadExternalPlugins() or refresh them in refreshPlugins().

Within refreshPlugins() on line 282, startPlugin() is called:

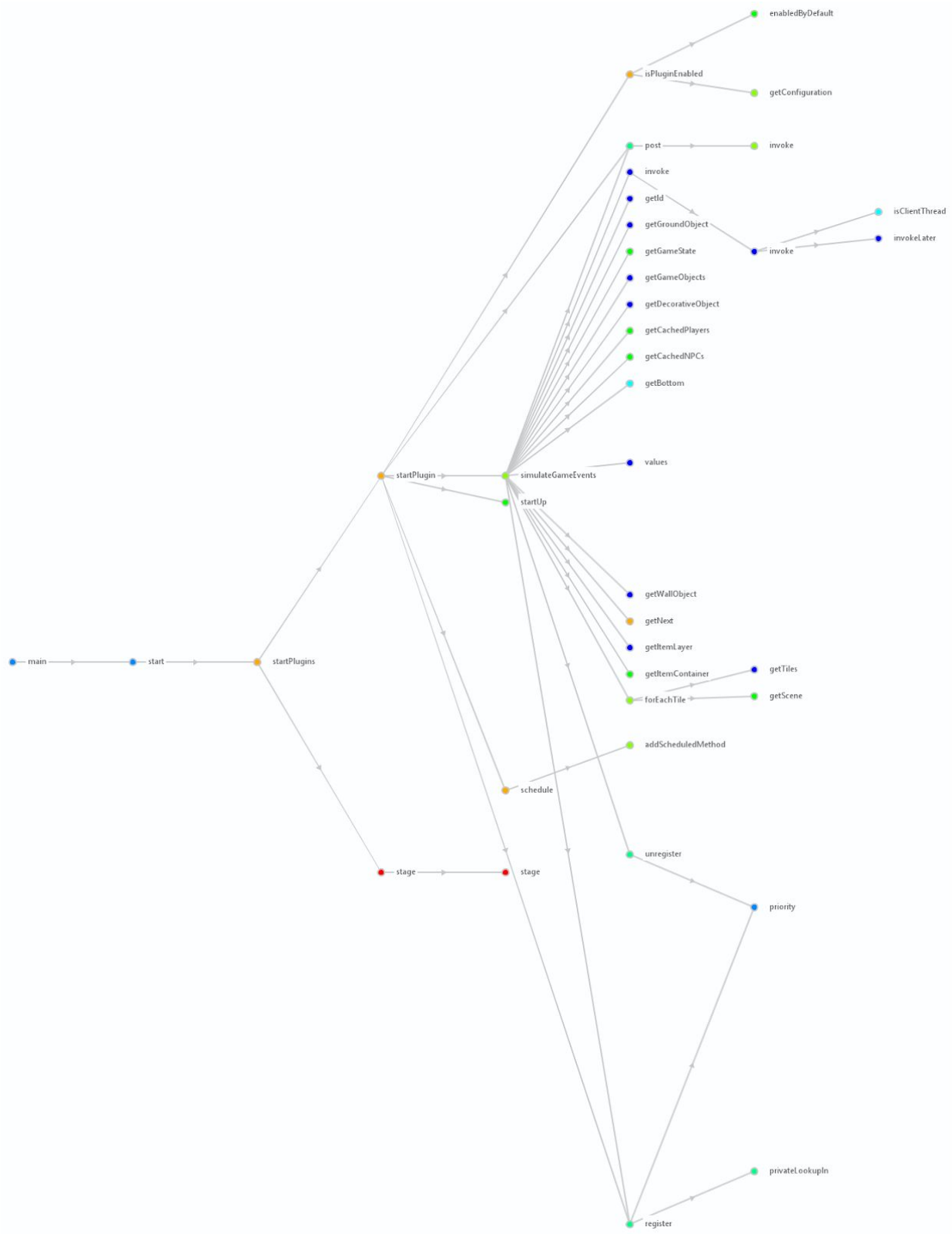
```
List<Plugin> newPlugins = null;
try
{
    ClassLoader cl = new ExternalPluginClassLoader(manifest, new URL[]{manifest.getJarFile().toURI()
    List<Class<?>> clazzes = new ArrayList<>();
    for (String className : manifest.getPlugins())
    {
        clazzes.add(cl.loadClass(className));
    }

    newPlugins = pluginManager.loadPlugins(clazzes,  onPluginLoaded: null);
    if (!startup)
    {
        pluginManager.loadDefaultPluginConfiguration(newPlugins);

        for (Plugin p : newPlugins)
        {
            pluginManager.startPlugin(p);
        }
    }
}
catch (Exception e)
```

These two classes, PluginManager and ExternalPluginManager are relevant to the management of plugins and should be highly considered if someone wants to make a change to how plugins are added, which ones are added, and which ones should be removed. The actual creation or development of a new plugin can be found in the following links:
https://www.osrsbox.com/blog/2018/08/10/writing-runelite-plugins-part-1-building/
https://www.osrsbox.com/blog/2018/08/12/writing-runelite-plugins-part-2-structure/
https://www.osrsbox.com/blog/2018/08/18/writing-runelite-plugins-part-3-config/
https://www.osrsbox.com/blog/2019/01/17/writing-runelite-plugins-part-4-overlays/

Although instances of PluginManager and ExternalPluginManager are referenced in Runelite.java, this class is the entry point to the system and simply uses the instances to load the plugins. Therefore, Runelite.java is not as relevant as the other two classes if the focus is to potentially change how plugins are managed.

Observe runelite-client/target/maven-status/maven-compiler-plugin/compile/default-compile/{inputFiles.lst,createdFiles.lst}. While neither receive any mention in the remaining source code, something implicitly reads one or both of these files show plugins in the game client. If you want to explicitly add another plugin, and PluginManager doesn't add to these files, you may have to add to one or both of these files.

Call Graph for PluginManager generated from startPlugins():



Sequence Diagram for PluginManager generated from startPlugins():