Portfolio description

JavaLab

In this lab an SPILocator.java has been added in Common. This SPILocator locates all services described in the META-INF folder and places them in a Dictonary which can be used to find services who implement a specific interface. So, for example Game.java in Core uses the SPILocator's dictionary to locate all IGamePluginService providers, IEntityProcessingService providers and IPostEntityProcessingService.

This is done by java.util.ServiceLoader which looks for service providers in the resource directory META-INF/services. For example: Player has two files inside META-INF (**dk.sdu.mmmi.cbse.common.services.IEntityProcessingService** & **dk.sdu.mmmi.cbse.common.services.IGamePluginService**).
The names of these files are important as it is the path of the interface the service implements.
Inside these files the class path to the implemented service is written. For example: Inside IGamePluginService is "dk.sdu.mmmi.cbse.playersystem.PlayerPlugin" written as it points to the class who implements it in the project.

NetBeansLab1

In this lab the netbeans module system has been used. This enables the use of org.openide.util.Lookup inside Game.java. This Class uses annotations inside classes who provides services. For example AsteroidControlSystem.java has an annotation @ServiceProvider(service = IEntityProcessingService.class) which enables Game.java inside Core to look up all service providers of IEntityProcessingService. This replaces the need for the files inside META-INF from JavaLab. However, for the Lookup API to know where to search it is a requirement that the startup module (application) has a dependency on all the modules. This makes application to load in the modules JAR files, which will enable the Lookup to look inside each of the JARs for the service providers via annotation.

Besides this Core has an Installer class instead of a Main method. This Class extends *ModuleInstall*. This is run when the module gets installed during runtime

NetBeansLab2

For the first time in this course it is possible to dynamically remove and install modules during runtime with the netbeans module system. This is accomplished by the inclusion of the module SilentUpdate. This module looks at an updates.xml file, which contains module tags. These tags include details on how to load the module.
By the use of the SilentUpdate module and updates.xml it's possible to have a running application without any dependencies on other modules.
In order for this to work. Every module must still include annotations and use org.openide.util.Lookup to load the modules SilentUpdate is finding. Also, every module must have a class which implements Installer.

OSGiLab

Another way to dynamically remove and install modules is with OSGi and in this case Apache's implementation Felix. There is two different ways to declare a module. The first method (the loosest way) is through META-INF and an osgi.bnd file. This way is called declarative.
The second method is more specific to Felix which is with an Activator class which implements BundleActivator class.
Both methods serve the same purpose, to register service providers which the framework will then pick up and in this project add to a list of services. Core has a core.xml file which tells Felix which methods to call when it finds service providers such as IGamePlugin and IEntityProcessingService. This method adds the found service provider to a list inside Game.java which can then be used to loop through and provide the game logic.