NOTE: *Code and Videos are given as links, on a public GitHub domain.*
- Video 1 is located in Lab3
- Video 2 is located in Lab4

Javalab (1/4)

Link to Code /Video: https://github.com/davil18/Exercises_Jan_1/tree/master/Lab1

ServiceLoader extension provides 2 key elements for a running program, DictionaryDemo and DictionaryServiceprovider. DictionaryDemo contains the main class that runs the program, checking for existing words within the library. DictionaryServiceProvider finds and uses Dictionary components and uses them to give a proper response. If the word in the dictionary is not found, it will notify the client. See figure 1.

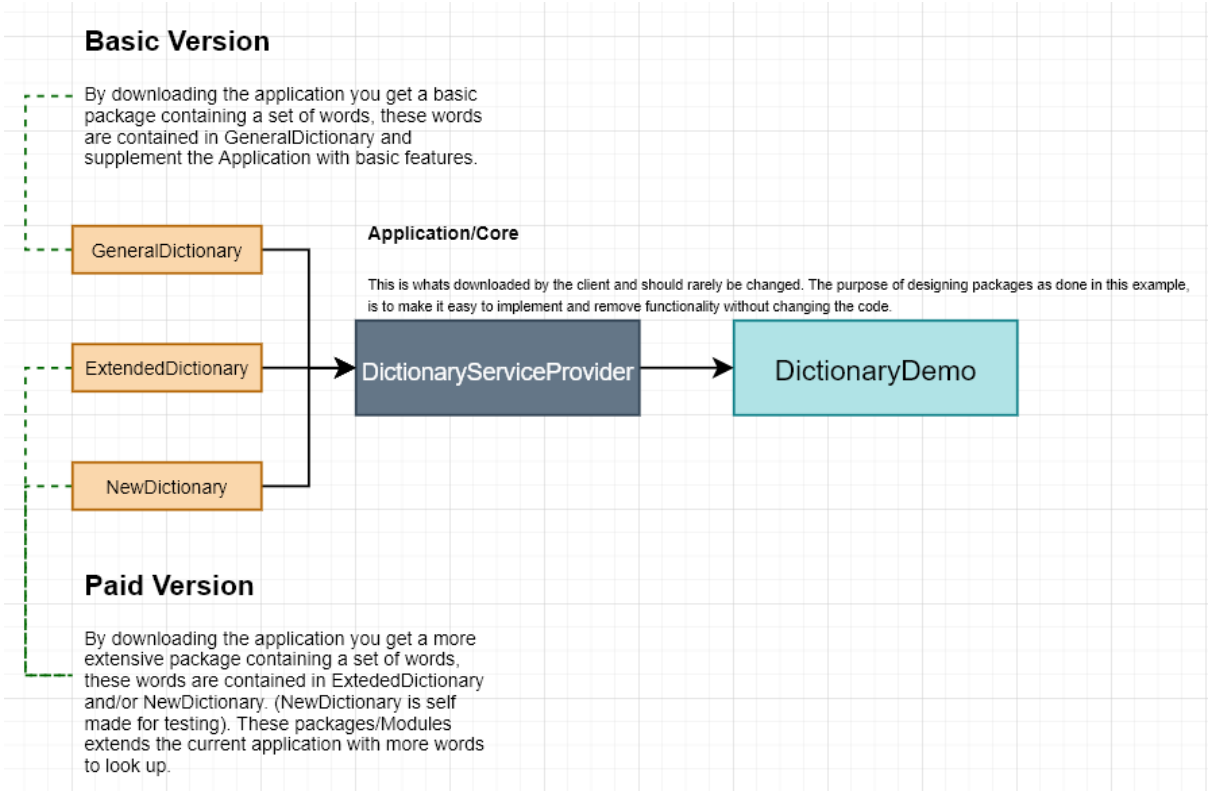

Figure 1 - ServiceLoaderExample

The purpose of this example is to show that you can implement new functionality without changing the source code, which can be beneficial for large projects. Implementing a supplementary package, the source code automatically read if the proper interfaces are implemented, is a lot easier than changing the source code every time. Simple implementing a package in parallel with other packages is an easy solution.

The interface serves as a key point to the module, it helps the DictionaryServiceProvider to locate which packages it should use, and whether it should use the package or not. Implementing a given interface to a module makes sure that the given module works as a Dictionary object, and it can be implemented as such, giving the DictionaryServiceProvider in this case, more words to look up.

NetBeansLab (2/4)

Link to Code/Video: https://github.com/davil18/Exercises_Jan_1/tree/master/Lab2

During this lab, the student was to be acquainted with NetBeans Platform, like the provided example AsteroidsNetbeansModules. Until now we have learned how to create object-oriented designs, focused on creating a program without modular pieces connected to it. We have created objects and classes within the same project, and imported .jar files as needed, but never really created them ourselves.

What this example shows us is that the modules (Player, Enemy, Collision, Bullet…etc) does not need to be in the same project to be implemented, instead be implemented dynamically as needed. This means that the main system or platform works without the individual components and can be added without directly changing the source code itself. See figure 2 for project and module separation.

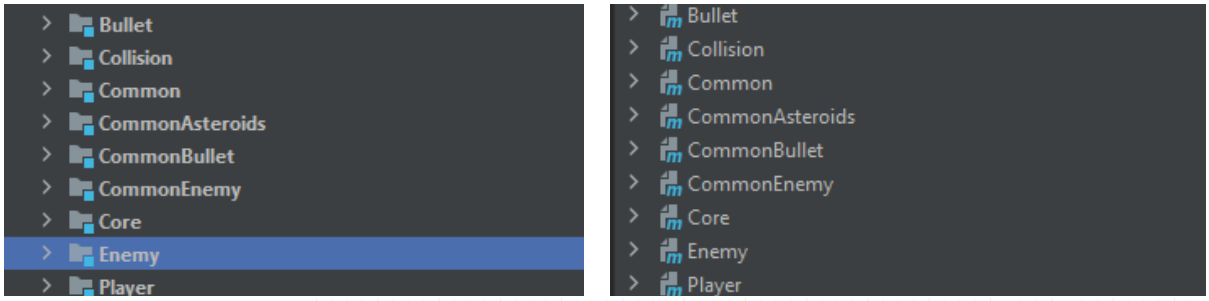

Figure 2

Usually the project would be singular, however we have now learned that we can implement whole projects as .jar files and use these projects as supplementary modules. A change that might be different from before, is access to these files from different packages. With my recent efforts into ServiceLoader, I learned that you need to make certain packages visible to certain packages, if they depend on that package. For example, if a class implements an interface in a different project, that class needs to see that interface for it to work. Making the interface public in that sense will make it available for said class, and that class can then use this interface as a supplementary module. See figure 3.

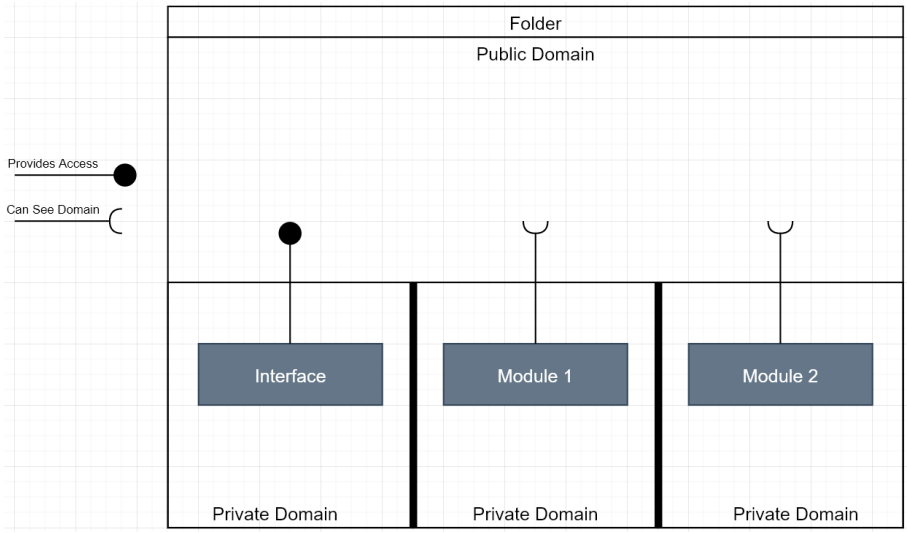

Figure 3

This illustration shows that if the interface has been made public, the modules can interact with it. When I tried to simply connect the dots by making the modules depend on the interface, it could not directly see it. This had to be changed in a file named **module-info.java**. At least in IntelliJ. Making to be able to use ServiceLoader, I then would need to create the same file type and specifically state that this module comes with the given interface. See figure 4. By doing this, ServiceLoader now can see these modules/implementations from different projects.

```
module ModPack1 {
    requires Interface;
    provides com.BeanInterface with com.Module1;
}
```

Figure 4

NetBeansLab (3/4)

Link to Code /Video: https://github.com/davil18/Exercises_Jan_1/tree/master/Lab3

In this exercise we were to implement modules during runtime. Meaning, that we could implement modules whilst the program was running, without having the dependency on the module itself. While I believe I successfully managed to do this following the instructions on the video, NetBeans still wanted to restart during runtime as I installed and uninstalled modules. See figure 5.

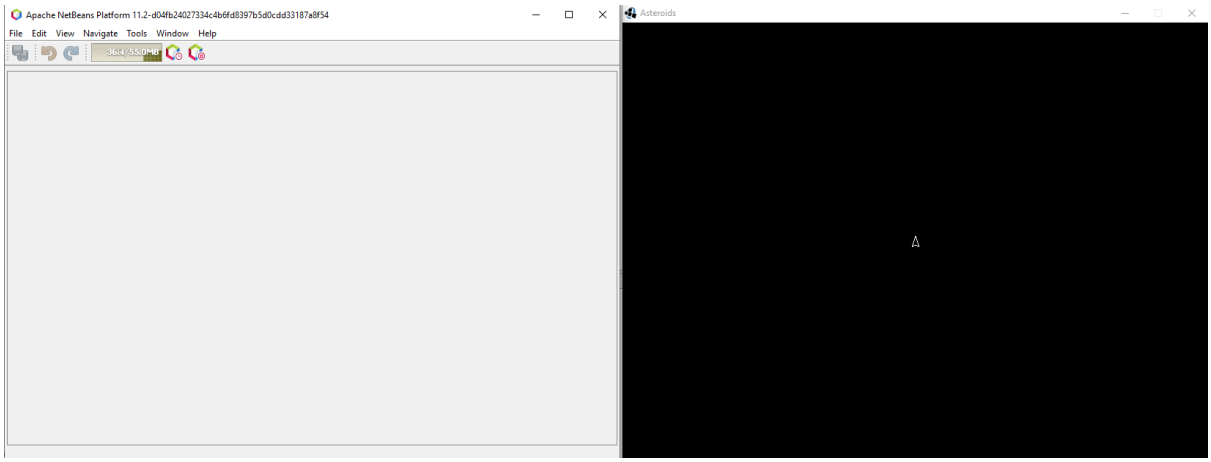

Figure 5

I could Activate/Deactivate and Uninstall/Install plugins at will, but it did not feel like it was changing this during its runtime, as NetBeans forced a restart after each Deactivation/Uninstall/Install of the module. This did not occur on Activation. Now that we have had continual problems throughout all these assignments, I did not investigate any further. See figure 6 for an overview of how the window looked for this process.

Creating the project itself was straight forward, however getting the service to register/see a variety of modules did not work as easily as hoped. Current feel of this solution is that it still requires a lot of on-hand work to get it to work with the source code itself, which makes the whole purpose of dynamic-loading in this case impractical. I have however, not following this process had great success on loading modules without even touching the source code itself, and I believe this area has great benefits with modern software-solutions.

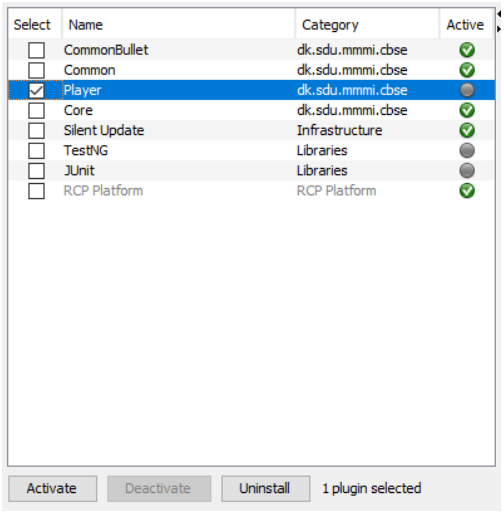

Figure 6

OSGi Lab (4/4)

Link to Code /Video: https://github.com/davil18/Exercises_Jan_1/tree/master/Lab4

OSGi is another method of implementing modules to your software. A difference from ordinary modules, bundles are introduced. Bundels being very similar to modules, introduces a manifest file located in ***META-INF/MANIFEST.MF***. Being like ServiceLoader, you need to declare the implementations of given modules as well as exporting the given module, if it is to be visible to other modules. Example from the book:

$$Import - Package: codabook.osgi.pos.model$$
$$Export - Package: codabook.osgi.pos.model$$

As mentioned above, you need to export a package to use it in another package, exporting and importing packages as shown above, will be done in the ***MANIFEST.MF*** file.

When this is done one can supposedly remove and add modules dynamically during runtime, with the provided example [PaxAsteroids] and the attempted implementation of OSGi network to our created asteroid game.

However, I did not manage to visibly implement the player module, I did however manage to implement the player bundle and make the program register its existence, implementing it with the Activator class. Even though I could not implement it visibly, it registered it as a separate bundle which could be deactivated and activated at will, which was the main goal. See figure 7.

Understanding how modules can be implemented during a running program can be very useful in the future.

```
-> mvn:dk.sdu.mmmi/OSGiPlayer/1.0-SNAPSHOT : connecting...
-> mvn:dk.sdu.mmmi/OSGiPlayer/1.0-SNAPSHOT : 8420 bytes @ [ 8420kBps ]

-> Using execution environment [J2SE-1.8]
-> Runner has successfully finished his job!
```

Figure 7

Lab 5, DesignLab & SpringLab

Coupling Table

AsteroidsLibGDX		
Class	DEPENDS ON	DEPENDENCY DEPTH
Player	SpaceObject, Game	2
SpaceObject	Game	1
GameState	GameStateManager	1
PlayState	GameState, GameStateManager, Player, GameKeys	4
Game	GameKeys, GameSatteManager, GameInputProcessor	3
Main	Game	1
GameInputProcessor	-	0
GameKeys	-	0
GameStateManager	-	0

AsteroidsNetbeansModules-parent		
Class	DEPENDS ON	DEPENDENCY DEPTH
Game	Entity, GameData, World, IEntityProcessingService, IGamePluginService, IPostEntityProcessingService, GameInputProcessor	7
World	Entity	1
Installer	-	0
AssetsJarFileResolver	-	0
GameInputProcessor	GameData, GameKeys	2
JarFileHandleStream	-	0
UpdateActivator	-	0
UpdateHandler	-	0
AsteroidPlugin	Asteroid, Entity, GameData, World, LifePart, MovingPart, PositionPart, IGamepLuginService	8
AsteroidProcessor	Asteroid, IAsteroidSplitter, Entity, GameData, World, LifePart, MovingPart, PositionPart, IEntityProcessingService	9
AsteroidSplitterImpl	Asteroid, IAsteroidSplitter, Entity, World, LifePart, MovingPart, PositionPart	7
Entity	EntityPart	1
EntityPart	Entity, GameData	2
LifePart	EntityPart, GameData, Entity	3
MovingPart	Entity, GameData, EntityPart	3
PositionPart	EntityPart, Entity, GameData	3

TimerPart	EntityPart, Entity, GameData, LifePart	4
Event	Entity	1
IEntityProcessingService	GameData, World	2
IGamePluginService	GameData, World	2
IPostEntityService	GameData, World	2
GameData	GameKeys, Event	2
GameKeys	-	0
Asteroid	Entity	1
IAsteroidSplitter	Entity, World	2
Enemy	Entity	1
Player	Entity	1
PlayerControlSystem	BulletSPI, Entity, GameData, GameKeys, World, LifePart, MovingPart, PositionPart, IEntityProcessingService	9
PlayerPlugin	Entity, GameData, World, LifePart, MovingPart, PositionPart, IGamePluginService	7
Bullet	Entity	1
BulletControlSystem	BulletSPI, Entity, GameData, World, LifePart, MovingPart, PositionPart, TimerPart, IEntityProcessingService	9
BulletPlugin	Entity, GameData, World, IGamePluginService, Bullet	5
CollisionDetector	World, GameData, Entity, LifePart, PositionPart, IPostEntityProcessingService	6
Bullet	Entity	1
BulletSPI	Entity, GameData	2

Reflection

One distinct difference between the two tables is the amount of content in one table compared to the other. Looking away from that, I see no real difference. The increase of modules or functionality also likely increases the dependency on other modules, which naturally increases the dependency on a single module. This means that there is high cohesion on specific parts, in this case Entity, which a variety of parts depend on to work. You could also remove LifePart to end the Player, Bullet and Asteroid at once. The idea behind making module-based systems is to decrease direct dependency on parts so they can work as independently as possible, restricting the uses of dependencies on other modules that might be removed. Doing this will create a chain reaction of errors that cause more damage than good. As for this example I can not see the direct difference, however, in relevance to our own project of recreating and modifying the asteroid game, one of our problems were that one of the new modules (Lets refer to this module as Module1) were directly dependant on literally every other module within the core of the game, which caused Module1 to fail as soon one of the other modules were missing. Consider having Module1 as a dependency on another module (Module2). Removing a single one of the core modules would cause the entire branch from Module1 and up to stop working or crash the entire program. This is worth noticing when creating large programs in large

projects. A toothpick might be able to “Hold-The-Door!”, but as soon that toothpick cracks everything will fall through.

Spring

Link to Code: https://github.com/davil18/Exercises_Jan_2/tree/master/AsteroidsEntityFramework

Primary differences between using Spring were the *.xml configuration files. While I got Spring up and running, similar to the given example AgeCalculatorSpring, I did not see the purpose of using it on module-based components. The key points I gathered from using the service compared to the old example, were as follows:

Without Spring	With Spring
Importing the module	*.xml configuration file
Declare the given class	Importing the module
	Declare the given class
	*.xml configuration file names needed to be unique pr. module
	Specify within the config file, which class you want to classify as a ‘bean’. `<bean id="getANumber" class="commie.gags.spring.GetANumber"/>`
	Import a class which handles the above: `ApplicationContext context = new ClassPathXmlApplicationContext("Bean.xml");`

While I at first were quite confused of how this were connected, I managed to get it running. It is fair to say, that during a previous semester, Spring became quite useful when we were creating endpoints to a given webservice. However, for component-based designs it is a mess. Creating an instance of a given object is much easier using ServiceLoader or not at all. Having to make sure that names are unique to a given instance of an *.xml file is just adding more trouble where there is none. I am sure, that there is a way to avoid this, but if Spring is supposed to make it easier to make component-based systems, it does not cut it. While I in the example without Spring could load all serviced that simply implemented an Interface, I instead now need to specify which classes I want to run in the *.xml file, for a given module and then specify within the module that have implemented said bean, what I want to use. I believe this will cause more cohesion in the long run and just chaos for the individual worker. Implementing an interface for a given module that can be put into a folder and work without touching the source code is much more effective at creating module-based systems.

Lab 6, TestLab

Testing, Junit

Link to Code: https://github.com/davil18/Exercises_Jan_2/tree/master/AsteroidsNetbeansModules

Testing components for correct outputs can help support the development of software, in this case large software projects. The usefulness of implementing these is to quickly see whether the output of code is similar or equal to the expected output. In this case, I chose to use the collision example, where collision between 2 entities takes place. See figure 1. The two tests made take a scenario of 2 entities with the added PositionParts(A class that specifies the central position of an Entity), used to validate whether the points of entity1 and entity2, are within a specific range of each other, this rage being 10 pixels. For the first test, the expected result is set to be true, because the 2 entities has a positional difference with 1px on X and Y, which is less than 10. By using the collision method set to 10 pixels, this should provide a result of true.

```
@org.junit.Test
public void testCollides() {
    Entity entity = new Entity();
    entity.add(new PositionPart(100, 100, 1));

    Entity entity2 = new Entity();
    entity2.add(new PositionPart(101, 101, 1));

    CollisionDetector instance = new CollisionDetector();
    Boolean expectedResult = true;
    Boolean result = instance.Collides(entity, entity2);

    assertEquals(expectedResult, result);
}

@org.junit.Test
public void testCollidesNot() {
    Entity entity = new Entity();
    entity.add(new PositionPart(100, 100, 1));

    Entity entity2 = new Entity();
    entity2.add(new PositionPart(151, 151, 1));

    CollisionDetector instance = new CollisionDetector();
    Boolean expectedResult = false;
    Boolean result = instance.Collides(entity, entity2);

    assertEquals(expectedResult, result);
}
```

Figure 8

For the second test, positions have a range difference of 50px on X and Y, which means that the test is expected to fail. See figure 2 and 3, to see how you start the test and see the result of the test shown in figure 1.

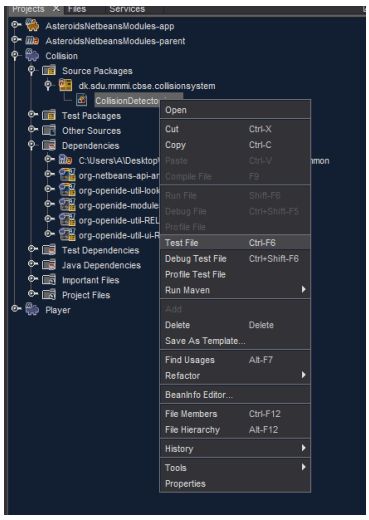

Figure 10

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running dk.sdu.mmmi.cbse.collisionsystem.CollisionDetectorTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.217 s - i

Results:

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.407 s
Finished at: 2021-04-20T18:44:19+02:00
------------------------------------------------------------------------
```

Figure 9

Reflection

While I find testing to be a useful tool, it should only be used if the code produced are to be worked on by other parties, or by junior developers. The development of tests does not take as long as creating the source code itself, but it does take time to create when more complex code is made. My preference in this area is to just create “Good Code” that are well structured and not messy, which will help developers understand what is happening, instead of relying on tests to complete that does the thinking for you, leaving you to guesswork.

Tests are great if the code is largely complex, and the code need to go quickly though the organizations branch of developers. But for the development of a new software where income itself are 0 on the current software being worked on, it is an expensive addition.