# 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.

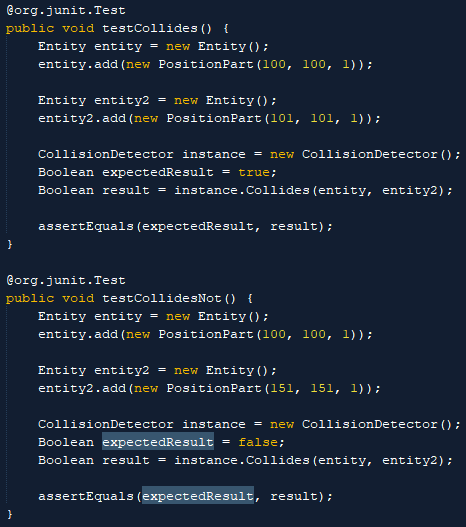


Figure 1

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.

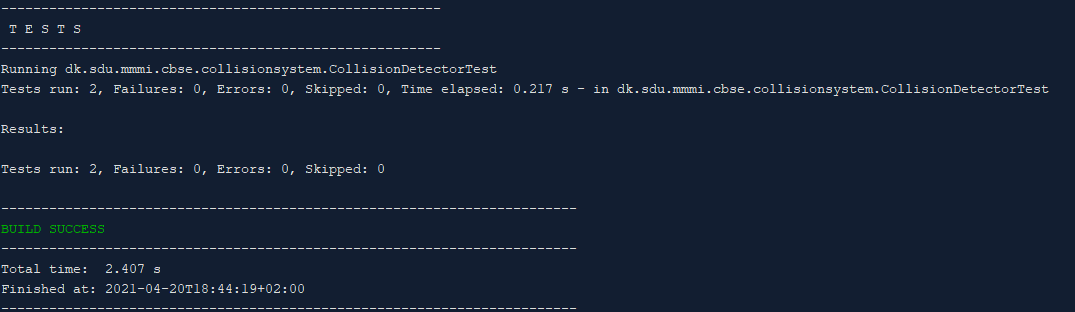


Figure 3

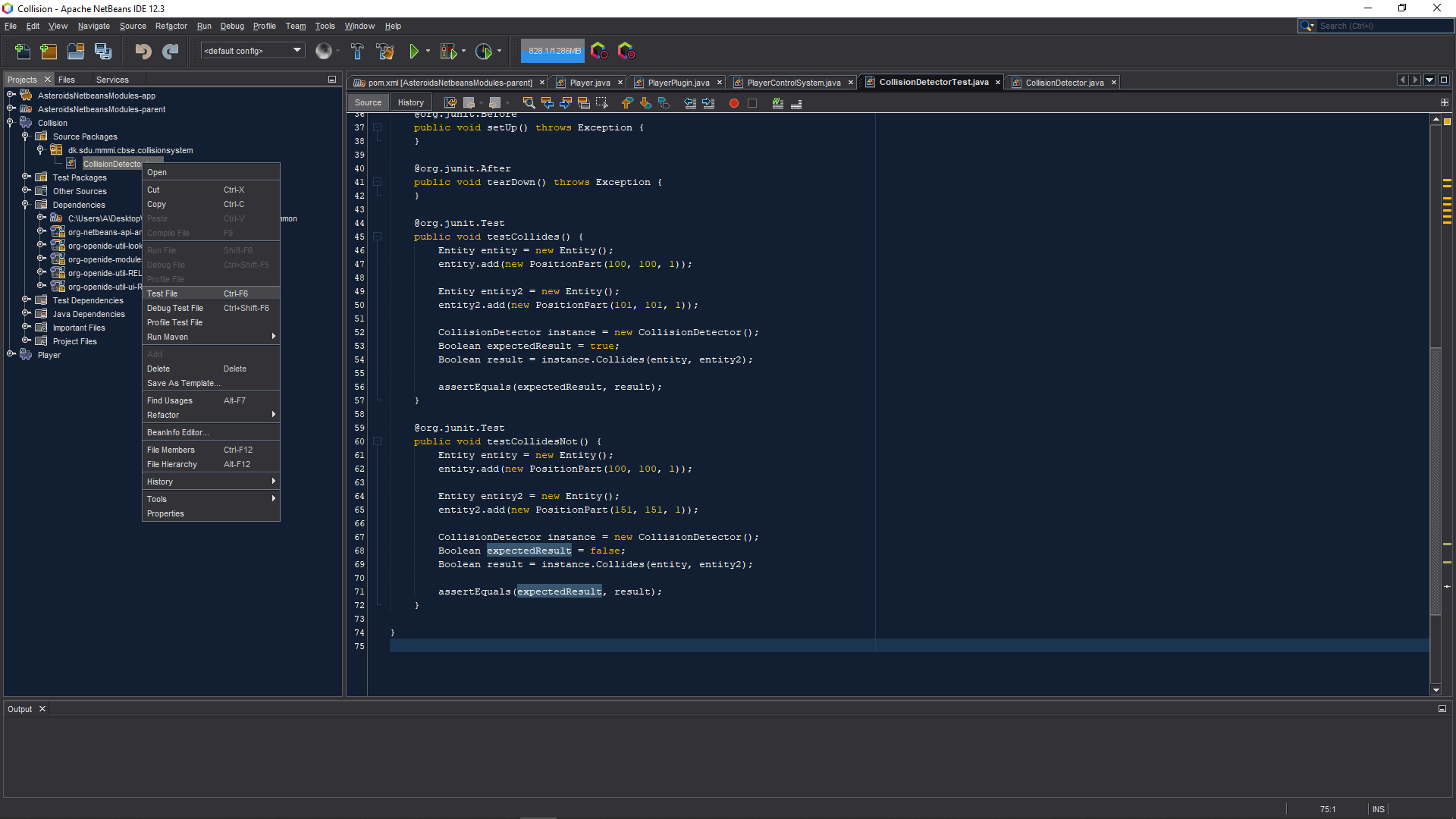


Figure 2

## 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.