Assignment Portfolio 1

Kevin K. M. Petersen Eksamens nr: 480900

May 9, 2021

1 Introduktion

Link til kildekode github.com/kepet19/komponent_systemer
Formålet med opgaverne er at komme igennem de forskellige komponent basseret arkitektur se fordele og ulemper ved dem. Finde ud af hvad forskellen er på Whiteboard Component Model og Dependency Injection og se om man finde ud af hvad de forskellige framework har implementeret.

1.1 Whiteboard Component Model

Whiteboard component model går ud på man registerer sit modul/klasse i et bestemt `Component Registry`.
I den component registry kan man så søge efter klasser som har implementeret et bestemt interface.
De klasser som component registry finder bliver så givet tilbage til spørgen. I den klasse hvor jeg har søgt i component registry giver mig så en klasse tilbage instantieret form.

1.2 Dependency Injection

Dependency injection er når en assembler står for at oprettet klasse og så enten sætte den ind på en anden klasse via den konstruktøren eller en sætter metode på klassen.

2 Javalab

I denne opgave skulle man lave et ekstra komponent som kunne loades ind via java service loader. Java service loader er en implementering af Whiteboard Component Model. I denne opgave lærer man hvordan man registere en klasse som kan blive loadet ind via java service loader.

2.1 Register komponent

først lave jeg en ny "Enemy" komponent som er baseret på player, da mange af tingende er de samme.
Derefter tilføjede jeg Enemy til `rodens/root` og `./pom.xml` filen, hvor den kom til at stå som et ekstra modul så maven vidste den skulle compiles med. Derefter tilføjede jeg "Enemy" til dependencies inde i "Core/pom.xml". så java kunne finde de interfaces som vi ledte efter. En af de key items er at man skal have en "resource" mappe med hvor er er en "META-INF" mappe og i den mappe har man en fil med navnet på det interface man siger man implementere. inde i filen siger man hvilke klasse der implementere dette interface.
Her har jeg en træ structur af hvordan filerne ser ud.

```
Enemy
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── dk.sdu.mmmi.cbse.enemysystem
    │           ├── Enemy.java
    │           ├── EnemyControlSystem.java
    │           └── EnemyPlugin.java
    └── resources
        └── META-INF
            ├── navent på det interface man siger man implementere
            ├── dk.sdu.mmmi.cbse.common.services.IEntityProcessingService
            └── dk.sdu.mmmi.cbse.common.services.IGamePluginService
```

Listing 1: dk.sdu.mmmi.cbse.common.services.IEntityProcessingService

```
1 dk.sdu.mmmi.cbse.enemysystem.EnemyControlSystem
```

""

Listing 2: dk.sdu.mmmi.cbse.common.services.igamepluginservice

```
1 dk.sdu.mmmi.cbse.enemysystem.enemyplugin
```

2.2 Loade modulerne ind via java ServiceLoader

I klassen Game som ligger i modulet `Core` bruger klassen der hedder `SPILocator` som ligger I modulet `Common`. `SPILocator` har en methode til at loade klasser ind med et bestemt interface. `SPILocator` implmentering kan ses i **Listing 3**. `SPILocator` bruger java standard service loader som kan ses på linje 20 I **Listing 3** til at loade de klasser som implementere det interface man har givet til serviceloaderen.

Java service loader søger efter et bestemt interface I jar filerne. Jar filerne inholder en mappe som hedder `META-INF/` hvor de så har en under mappe der hedder `services/`. I den mappe søger java service loader så efter et filnavn med det fulde interface navn(pakken er også med).

Listing 3: dk.sdu.mmmi.cbse.common.util.SPILocator.java

```
1  package dk.sdu.mmmi.cbse.common.util;
2
3  import java.util.*;
4
5  public class SPILocator {
6
7      @SuppressWarnings("rawtypes")
8      private static final Map<Class, ServiceLoader> loadermap = new HashMap<
           Class, ServiceLoader>();
9
10     private SPILocator() {
11     }
12
13     @SuppressWarnings("unchecked")
14     public static <T> List<T> locateAll(Class<T> service) {
15         ServiceLoader<T> loader = loadermap.get(service);
16
17         boolean printStatement = false;
18
19         if (loader == null) {
20             loader = ServiceLoader.load(service);
21             loadermap.put(service, loader);
22             printStatement = true;
23         }
24
25         List<T> list = new ArrayList<T>();
26
27         if (loader != null) {
28             try {
29                 for (T instance : loader) {
30                     list.add(instance);
31                 }
32             } catch (ServiceConfigurationError serviceError) {
33                 serviceError.printStackTrace();
34             }
35         }
36
37         if (printStatement) {
38             System.out.println("Found " + list.size() + " implementations
                   for interface: " + service.getName());
39         }
40
41         return list;
42     }
43 }
```

3 NetBeansLab1

I denne ogpave skal man lave et netbean module som kan loades ind via "Netbeans module system" som er en implementering af Whiteboard Component Model. Der er allered givet et example projekt, som jeg har udvidet på.

3.1 Register komponent

For at registere et komponent skal man først lave et netbeans module som skal lægges ind som dependecy inde i "application/pom.xml", så netbeans module system ved at den skal loade modulet som der er blevet lavet.

```
Astroid6Shapes
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── dk.sdu.mmmi.cbse.astroid6shapes
    │           ├── AsteroidPlugin.java
    │           ├── AsteroidProcessor.java
    │           └── AsteroidSplitterImpl.java
    ├── nbm
    │   └── manifest.mf
    └── resources
        └── dk.sdu.mmmi.cbse.astroid6shapes
            └── Bundle.properties
```

I **Listing 4** ses implementation på klassen **AsteroidPlugin.java**. På linje 6 ses en decorator der fortæller netbeanas module system at dette er en klasse som provider IGamePluginService interfacet.

Listing 4: AsteroidPlugin.java

```
1  package dk.sdu.mmmi.cbse.astroid6shapes;
2
3  import dk.sdu.mmmi.cbse...*;
4  import org.openide.util.lookup.ServiceProvider;
5
6  @ServiceProvider(service = IGamePluginService.class)
7  public class AsteroidPlugin implements IGamePluginService{
8          @Override
9      public void start(GameData gameData, World world) {
10         ...
11     }
12
13     @Override
14     public void stop(GameData gameData, World world) {
15                 world.getEntities(Astroid6shapes.class).forEach(asteroid ->
                       {
16                         world.removeEntity(asteroid);
17                 });
18     }
19 }
```

3.2 Netbeans module system lille og stor

Netbeans module system har 2 forskellige systemer en slim uden noget grafisk, kun til at loade netbeans modules med og en stor hvor man får netbeans grafiske interface med.

3.3 Loade modulerene ind via Netbeans

Loade modulerne ind via netbeans module system sker ved netbeans Lookup api. Man kan se dem med interfacet `IGamePluginService` bliver loadet i **Listing 5** mellem linje 45 til 52.

Listing 5: dk.sdu.mmmi.cbse.core.main.Game.java

```
22 public class Game implements ApplicationListener {
23
24     private static OrthographicCamera cam;
25     private ShapeRenderer sr;
26     private final Lookup lookup = Lookup.getDefault();
27     private final GameData gameData = new GameData();
28     private World world = new World();
29     private List<IGamePluginService> gamePlugins = new CopyOnWriteArrayList
           <>();
30     private Lookup.Result<IGamePluginService> result;
31
32     @Override
33     public void create() {
34         gameData.setDisplayWidth(Gdx.graphics.getWidth());
35         gameData.setDisplayHeight(Gdx.graphics.getHeight());
36
37         cam = new OrthographicCamera(gameData.getDisplayWidth(), gameData.
               getDisplayHeight());
38         cam.translate(gameData.getDisplayWidth() / 2, gameData.
               getDisplayHeight() / 2);
39         cam.update();
40
41         sr = new ShapeRenderer();
42
43         Gdx.input.setInputProcessor(new GameInputProcessor(gameData));
44
45         result = lookup.lookupResult(IGamePluginService.class);
46         result.addLookupListener(lookupListener);
47         result.allItems();
48
49         for (IGamePluginService plugin : result.allInstances()) {
50             plugin.start(gameData, world);
51             gamePlugins.add(plugin);
52         }
53     }
54
55     @Override
56     public void render() {
57         // clear screen to black
58         Gdx.gl.glClearColor(0, 0, 0, 1);
59         Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
60
61         gameData.setDelta(Gdx.graphics.getDeltaTime());
62         gameData.getKeys().update();
63
64         update();
65         draw();
```

```
66      }
67
68      private void update() {
69          // Update
70          for (IEntityProcessingService entityProcessorService :
                getEntityProcessingServices()) {
71              entityProcessorService.process(gameData, world);
72          }
73
74          // Post Update
75          for (IPostEntityProcessingService postEntityProcessorService :
                getPostEntityProcessingServices()) {
76              postEntityProcessorService.process(gameData, world);
77          }
78      }
79
80      private void draw() {
81          for (Entity entity : world.getEntities()) {
82              sr.setColor(1, 1, 1, 1);
83
84              sr.begin(ShapeRenderer.ShapeType.Line);
85
86              float[] shapex = entity.getShapeX();
87              float[] shapey = entity.getShapeY();
88
89              for (int i = 0, j = shapex.length - 1;
90                      i < shapex.length;
91                      j = i++) {
92
93                  sr.line(shapex[i], shapey[i], shapex[j], shapey[j]);
94              }
95
96              sr.end();
97          }
98      }
99
100     @Override
101     public void resize(int width, int height) {
102     }
103
104     @Override
105     public void pause() {
106     }
107
108     @Override
109     public void resume() {
110     }
111
112     @Override
113     public void dispose() {
114     }
115
116     private Collection<? extends IEntityProcessingService>
            getEntityProcessingServices() {
117         return lookup.lookupAll(IEntityProcessingService.class);
118     }
119
120     private Collection<? extends IPostEntityProcessingService>
            getPostEntityProcessingServices() {
```

```
121         return lookup.lookupAll(IPostEntityProcessingService.class);
122     }
123 
124     private final LookupListener lookupListener = new LookupListener() {
125         @Override
126         public void resultChanged(LookupEvent le) {
127 
128             Collection<? extends IGamePluginService> updated = result.
                    allInstances();
129 
130             for (IGamePluginService us : updated) {
131                 // Newly installed modules
132                 if (!gamePlugins.contains(us)) {
133                     us.start(gameData, world);
134                     gamePlugins.add(us);
135                 }
136             }
137 
138             // Stop and remove module
139             for (IGamePluginService gs : gamePlugins) {
140                 if (!updated.contains(gs)) {
141                     gs.stop(gameData, world);
142                     gamePlugins.remove(gs);
143                 }
144             }
145         }
146 
147     };
148 }
```

4 NetBeansLab2

Netbeans lab 2 skal man vise at netbeans module system kan loade og undeloade moduler i runtime/kørslen af programmet. Der der allerede er lavet et modul "SilentUpdate" som kan søger for at installere og afinstallér moduler i runtime. Det modul smider man så som en dependecy inde i "application/pom.xml" filen.

4.1 Register komponent

Denne her gang fjerner jeg det module jeg lavede i netbeans lab 1 "Astroid6Shapes" dependecy fra "application/pom.xml" for at vise den kan bliver loadet ind med SilentUpdate.

```
SilentUpdate
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── org.netbeans.modules.autoupdate.silentupdate
    │           ├── UpdateActivator.java
    │           └── UpdateHandler.java
    ├── nbm
    │   └── manifest.mf
    └── resources
        └── org.netbeans.modules.autoupdate.silentupdate
            └── resources
                ├── Bundle.properties
                └── layer.xml
```

I filen "Bundle.properties" skal man pege på ens update center. jeg har valgt at bruge "target" mappen i application mappen som "update site".
Her under ses den bundle.properties. Linje 7 hvor man kan ændre hvor "netbeans site" peger til. Jeg har dog valgt at ændre navnet til kort et. da det ikke ville værer pænt med full path til filen.

Listing 6: Bundle.properties

```
1 #Tue Mar 10 14:13:59 CET 2015
2 Services/AutoupdateType/
    org_netbeans_modules_autoupdate_silentupdate_update_center.instance=
    Sample Update Center
3 OpenIDE-Module-Display-Category=Infrastructure
4 OutputLogger.Grain=VERBOSE
5 OpenIDE-Module-Name=Silent Update
6 OpenIDE-Module-Short-Description=Silent Update of Application
7 org_netbeans_modules_autoupdate_silentupdate_update_center=file:///target/
    netbeans_site/updates.xml
8 OpenIDE-Module-Long-Description=A service installing updates of your
    NetBeans Platform Application with as few as possible user's interactions
    .
```

4.2 Netbeans site

Netbeans site modulerne består af information om hvad de vil have, hvad de giver af service til andre og de indeholder også java byte code (jar filer).

4.3 Komponentdet bliver loadet og undloadet via silent update

Se video: youtu.be/H2Y0cPQ0fzc

5 OSGiLab

I OSGI lab skal der laves 2 ny moduler, fordi der skal laves en med OSGi Declarative Services(Dependency injection) og en med OSGi BundeContext API(Whiteboard komponent modellen).

5.1 OSGi Declarative Services

I OSGi declarative services bruger vi en "osgi.bnd" til at fortælle OSGi hvilke activationpolicy der skal værer. Den fortæller også noget om hvor den kan finde noget om de service der bliver leveret fra modulet se **Listing 7**.

```
Player
├── osgi.bnd
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── dk.sdu.mmmi.cbse.playersystem
    │           ├── Player.java
    │           ├── PlayerControlSystem.java
    │           └── PlayerPlugin.java
    └── resources
        └── META-INF
            ├── entityprocessor.xml
            └── gameplugin.xml
```

Listing 7: osgi.bnd

```
1 Bundle-SymbolicName: Player
2 Bundle-ActivationPolicy: lazy
3 Service-Component: META-INF/entityprocessor.xml, META-INF/gameplugin.xml
```

I **Listing 8** ses på linje 3 at det er implementation klassen som den peger på og den implementere interfaces som er listest på linje 5.

Listing 8: gameplugin.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="dk.sdu
    .mmmi.cbse.playersystem.plugin">
3     <implementation class="dk.sdu.mmmi.cbse.playersystem.PlayerPlugin"/>
4     <service>
5         <provide interface="dk.sdu.mmmi.cbse.common.services.
              IGamePluginService"/>
6     </service>
7 </scr:component>
```

5.2 OSGI BundleContext API

```
Enemy2
├── pom.xml
└── src
    └── main
        └── java
            └── dk.sdu.mmmi.cbse.osgienemy2
                ├── Activator.java
                ├── Enemy2.java
                ├── Enemy2Plugin.java
                └── Enemy2Processor.java
```

5.3 Register komponent

I **Listing 9** ses den pom.xml som peger på den klasse som har implementeret bundle activator.

Listing 9: pom.xml

```
1     <build>
2         <plugins>
3             <plugin>
4                 <groupId>org.apache.felix</groupId>
5                 <artifactId>maven-bundle-plugin</artifactId>
6                 <extensions>true</extensions>
7                 <configuration>
8                     <instructions>
9                         <Bundle-Activator>dk.sdu.mmmi.cbse.osgienemy2.
                              Activator</Bundle-Activator>
10                    </instructions>
11                </configuration>
12            </plugin>
13        </plugins>
14    </build>
```

I **Listing 10** ses den klasse som har implementeret bundle activator.

Listing 10: Activator.java

```
1  package dk.sdu.mmmi.cbse.osgienemy2;
2
3  import dk.sdu.mmmi.cbse.common.services.IEntityProcessingService;
4  import dk.sdu.mmmi.cbse.common.services.IGamePluginService;
5  import org.osgi.framework.BundleActivator;
6  import org.osgi.framework.BundleContext;
7
8  public class Activator implements BundleActivator {
9
10     @Override
11     public void start(BundleContext context) throws Exception {
12
13         context.registerService(IGamePluginService.class, new Enemy2Plugin()
               , null);
14         context.registerService(IEntityProcessingService.class, new
               Enemy2Processor(), null);
```

```
15      }
16
17      public void stop(BundleContext context) throws Exception {
18      }
19
20  }
```

5.4 Komponentdet bliver loadet og unloadet via Apache gogo shell

Se video youtu.be/nB84hmut8W0

6 DesignLab

6.1 opgave 1

I denne opgave skal man tælle afhængighed dybde(dependency depth) i det monolitisk spil Asteroids og I det modul baseret spil Asteroids **[NetbeansLab2]**
I **Tablen 1** sidste kolonne er der dependency depth ved nogle af tallende står der en "*" astrict betyder den er den er cyclist dependency. Hvis vi tager udgangs punkt i klassen `PlayState` kan man se fra den første dependecy `GameStateManager` at den referere tilbage til `PlayState` og `GameStateManager` har også en dependency på `GameState` referere direkte tilbage til `GameStateManager`. Den vej jeg har fundet som er mest dyb uden at gå i samme dependency er `Player>SpaceObject>Game>GameKeys` fra `PlayState`.
Ud fra **Table 1** og **Table 2** har jeg konkluderet at det er nemmere at villigeholde i den komponent baseret spil da der er en meget lille kobling.

Table 1: List of dependency depth for monolithic Asteroids Game

NR	CLASS	DEPENDS ON	DEPENDENCY DEPTH
1	GameKeys		0
2	GameInputProcessor	GameKeys	1
3	PlayState	GameStateManager, GameState, Player, GameKeys	4*
4	GameStateManager	GameState, PlayState	5*
5	Game	GameStateManager, Player, GameKeys, GameInputProcessor	5*
6	SpaceObject	Game	5*
7	Player	SpaceObject, Game, PlayState	5*
8	GameState	GameStateManager	6*
9	Main	Game	6*

Table 2: List of dependency depth for module based Asteroids Game

NR	CLASS	DEPENDS ON	DEPENDENCY DEPTH
1	Common		0
2	SilentUpdate		0
3	CommonBullet	Common	1
4	CommonEnemy	Common	1
5	CommonAsteroids	Common	1
6	Collision	Common	1
7	Core	Common	1
8	Player	Common, CommonBullet	2
9	Bullet	Common, CommonBullet	2
10	Enemy	Common, CommonEnemy	2
11	Asteroid	Common, CommonAsteroids	2
12	Asteroid6Shapes	Common, CommonAsteroids	2

6.2 Fordele og ulemper ved komponent baseret design

Hvad er fordele og ulemper ved komponent baseret design?
Fordele:

- Man for som regel løs kobling og høj samhørlighed (low coupling and high cohesion).
- Mange gange for man kode som er nemmere at villigeholde og udskifte på komponenterne.

Ulemper:

- Det kan tage længere tid at designe da man skal finde ud af hvor man skal sætte grænserne for komponenterne.
- Hvis man designer noget forkert kan det værre svært at ændre komponent interface, da der måske er flere komponenter der allerede er afhængige af det.

7 SpringLab

I denne opgave skal man porte **[JavaLab]** til at være basseret på Spring framworket. Det har jeg gjort ved først at kigge på **[SpringAgeCalculator]** for at få en ide om hvordan det virker.
Derefter åbenede jeg project fra **[JavaLab]** og fandt ud af hvilke moduler som skulle depency injectes via spring frameworket. Dem der skulle konventeres er:

- **Asteroid**
- **Collision**
- **Enemy**
- **Player**

7.1 Enemy modulet

Jeg tager udgangspunkt I **Enemy**. Fil træet for enemy ser sådan ud.

```
Enemy
├── src
│   ├── main
│   │   ├── java
│   │   │   └── dk.sdu.mmmi.cbse.enemysystem
│   │   │       ├── Enemy.java
│   │   │       ├── EnemyControlSystem.java
│   │   │       └── EnemyPlugin.java
│   │   └── resources
│   │       └── enemyBean.xml
└── pom.xml
```

Det vigtigste der har ændret sig i filen strukturen er der er kommet en ny fil(**enemyBean.xml**) der beskriver et id på en bean og hvor man kan finde den. Jeg har også slettet mappen **META-INF** der blev brugt til Java's service loader.
enemyBean.xml kan ses i **Listing 11**. Beans id'erne skal bruges i den bean hvor jeg samler det hele (kommer senere).

Listing 11: enemyBean.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans>
3     <bean id="enemyControlSystem" class="dk.sdu.mmmi.cbse.enemysystem.
          EnemyControlSystem"/>
4     <bean id="enemyPlugin" class="dk.sdu.mmmi.cbse.enemysystem.EnemyPlugin"/
          >
5
6 </beans>
```

7.2 Core modulet

I **Core** modulet skal der en ny dependency in i pom.xml. filen kan ses i **Listing 12**. Der kommer jeg spring frameworket dependency'en ind.

Listing 12: core:pom.xml

```
92          <dependency>
93              <groupId>org.springframework</groupId>
94              <artifactId>spring-context</artifactId>
95              <version>4.1.5.RELEASE</version>
96          </dependency>
```

Derefter opretter jeg en ny fill der hedder **Beans.xml** den står så for at beskrive hvordan bean ser ud i filen. Bean id'et er "core" og klassen er Game. I Beans.xml filen importer jeg også de andre xml filer. På linje 6 i **Listing 13** kan man se at den tidligere beskrevet enemyBean.xml bliver importeder her.
I den bean man er i gang med at beskrive(core) er der en property. Den property har et navn som skal være tilgændelig som en sætter methode i Game klassen før spring kan dependency injecte de andre klasser. Property'en har en liste med "value-type" som siger at listen er af dette interface og pejer på interfacet klassen. klasserne som implementere det tidligere nævnte interface som man gerne vil have dependency injectet skal man putte i listen. Listen består af id'er på de andre beans.

```
Core
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── dk.sdu.mmmi.cbse
    │           ├── main
    │           │   ├── Game.java
    │           │   └── Main.java
    │           └── managers
    │               └── GameInputProcessor.java
    └── resources
        └── Beans.xml
```

Listing 13: Beans.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans>
3      <import resource="asteroidsBean.xml" />
4      <import resource="collisionBean.xml" />
5      <import resource="playerBean.xml" />
6      <import resource="enemyBean.xml" />
7
8      <bean id="core" class="dk.sdu.mmmi.cbse.main.Game">
9          <property name="entityProcessors">
10             <list value-type="dk.sdu.mmmi.cbse.common.services.
                   IEntityProcessingService">
11                 <ref bean="playerControlSystem"/>
12                 <ref bean="asteroidControlSystem"/>
13                 <ref bean="asteroidSplitter"/>
14                 <ref bean="enemyControlSystem"/>
15
16             </list>
17         </property>
18         <property name="postEntityProcessors">
19             <list value-type="dk.sdu.mmmi.cbse.common.services.
```

```
                IPostEntityProcessingService">
20                <ref bean="collider"/>
21                <ref bean="asteroidPlugin"/>
22            </list>
23        </property>
24        <property name="pluginServices">
25            <list value-type="dk.sdu.mmmi.cbse.common.services.
                  IGamePluginService">
26                <ref bean="playerPlugin"/>
27                <ref bean="asteroidPlugin"/>
28                <ref bean="enemyPlugin"/>
29            </list>
30        </property>
31    </bean>
32
33 </beans>
```

I **Listing 14** ses der er lavet endnu en liste med gamePlugin.

Listing 14: dk.sdu.mmmi.cbse.main.Game.java

```
13 public class Game
14         implements ApplicationListener {
15
16     private static OrthographicCamera cam;
17     private ShapeRenderer sr;
18
19     private final GameData gameData = new GameData();
20     private List<IEntityProcessingService> entityProcessors = new ArrayList
           <>();
21     private List<IPostEntityProcessingService> postEntityProcessors = new
           ArrayList<>();
22     private List<IGamePluginService> gamePlugin = new ArrayList<>();
```

I **Listing 15** ses der er lavet 3 methoder så spring kan dependency injecte klasserne.

Listing 15: dk.sdu.mmmi.cbse.main.Game.java

```
109     public void setPluginServices(List<IGamePluginService> gamePlugin) {
110         this.gamePlugin = gamePlugin;
111     }
112
113     public void setPostEntityProcessors(List<IPostEntityProcessingService>
114             postEntityProcessors) {
115         this.postEntityProcessors = postEntityProcessors;
116     }
117
118     public void setEntityProcessors(List<IEntityProcessingService>
119             entityProcessors) {
120         this.entityProcessors = entityProcessors;
121     }
122 }
```

For at oprette et game objecet/klasse med spring framworket skal man importere spring framworket ClassPathXmlApplicationContext. Kalde konstrukteren på ClassPathXmlApplicationContext med parameteret på den Bean.xml man har lavet. Efter at have fået applicationContex fra ClassPathXmlApplicationContext kan man få selve bean ud ved at kalde methoden `getBean()` med et parameter id. Id'et er fra den bean man gerne vil have. Vi kunne godt tænke us den fra core som er game klassen. Det kan ses i **Listing 16** på linje 10. Der caster vi den så til et Game object,

så den kan bruges normalt. Spring frameworket har sørget for at gører de dele som er beskrevet i `Beans.xml` og dependency inject de klasser som er beskrevet.

Listing 16: dk.sdu.mmmi.cbse.main.Main.java

```
1  package dk.sdu.mmmi.cbse.main;
2
3  import com.badlogic.gdx.backends.lwjgl.*;
4  import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6  public class Main {
7
8      public static void main(String[] args) {
9          var context = new ClassPathXmlApplicationContext("Beans.xml");
10         Game game = (Game) context.getBean("core");
11
12         LwjglApplicationConfiguration cfg =
13         new LwjglApplicationConfiguration();
14         cfg.title = "Asteroids";
15         cfg.width = 500;
16         cfg.height = 400;
17         cfg.useGL30 = false;
18         cfg.resizable = false;
19
20         new LwjglApplication(game, cfg);
21
22     }
23
24     }
```

8 TestLab

I denne opgave skal vi lave testing, det har jeg gjort på 2 forskellige måder. Jeg har valgt at lave testing udfra **[JavaLab]** og jeg tester på klassen `PlayerControlSystem` som ligger i `Player` modulet.

I **@Test 1** som man kan i I **Listing 17** Der tester jeg uden Mockite for at vise hvor svært det er at teste en function uden at sætte en hel masse op først.
I **@Test 2** som man også kan se i **Listing 17** bruger jeg `Mockito` testing lib. Det gør det nemt at teste at functioner bliver kaldt inde i en function.

Listing 17: dk.sdu.mmmi.cbse.playersystem.PlayerControlSystemTest.java

```
1  package dk.sdu.mmmi.cbse.playersystem;
2
3  import java.util.ArrayList;
4  import static org.mockito.Mockito.*;
5  import static org.junit.jupiter.api.Assertions.*;
6  import org.junit.jupiter.api.Test;
7
8  import dk.sdu.mmmi.cbse.common.data.*;
9  import dk.sdu.mmmi.cbse.common.data.entityparts.*;
10
11 public class PlayerControlSystemTest {
12
13     @Test //Test 1
14     public void process() {
15         World world = new World();
16         GameData gameData = new GameData();
17         gameData.setDisplayHeight(400);
18         gameData.setDisplayWidth(500);
19         gameData.getKeys().setKey(GameKeys.UP, true);
20         new PlayerPlugin().start(gameData, world);
21
22         Entity player = world.getEntities(Player.class).get(0);
23         PositionPart positionPart = player.getPart(PositionPart.class);
24
25         // Checker om den start samme sted, som det der er programeet.
26         // Det gør jeg fordi så kan jeg teste om process rykker
27         // spilleren(player'en)
28         assertEquals(gameData.getDisplayWidth() / 2 , positionPart.getX());
29         assertEquals(gameData.getDisplayHeight() / 2 , positionPart.getY());
30         gameData.setDelta(0.5f);
31         new PlayerControlSystem().process(gameData, world);
32         // Efter jeg har kørt methode "process" så skulle den gerne have
               rykket
33         // sig så den er ikke ligemed.
34         assertNotEquals(gameData.getDisplayWidth() / 2, positionPart.getX())
               ;
35         assertNotEquals(gameData.getDisplayHeight() / 2, positionPart.getY()
               );
36
37
38
39     }
40
41     @Test //Test 2
42     public void TestThattheRighMethodsGetsCalled() {
43         World world = mock(World.class);
```

```
44
45          MovingPart mov = mock(MovingPart.class);
46          PositionPart pos = mock(PositionPart.class);
47
48          Player player = mock(Player.class);
49          when(player.getPart(PositionPart.class)).thenReturn(pos);
50          when(player.getPart(MovingPart.class)).thenReturn(mov);
51
52          when(player.getShapeX()).thenReturn(new float[4]);
53          when(player.getShapeY()).thenReturn(new float[4]);
54
55          ArrayList<Entity> listPlayer = new ArrayList<Entity>();
56          listPlayer.add(player);
57          when(world.getEntities(Player.class)).thenReturn(listPlayer);
58
59          GameData gameData = mock(GameData.class);
60          when(gameData.getKeys()).thenReturn(new GameKeys());
61
62
63          PlayerControlSystem pcs = new PlayerControlSystem();
64
65          pcs.process(gameData, world);
66          // Her checker den kalder methoderne på MovingPart og PositionPart
67          verify(mov).process(gameData, player);
68          verify(pos).process(gameData, player);
69          // Her checker jeg om den kalder get entries.
70          verify(world).getEntities(Player.class);
71      }
72  }
```