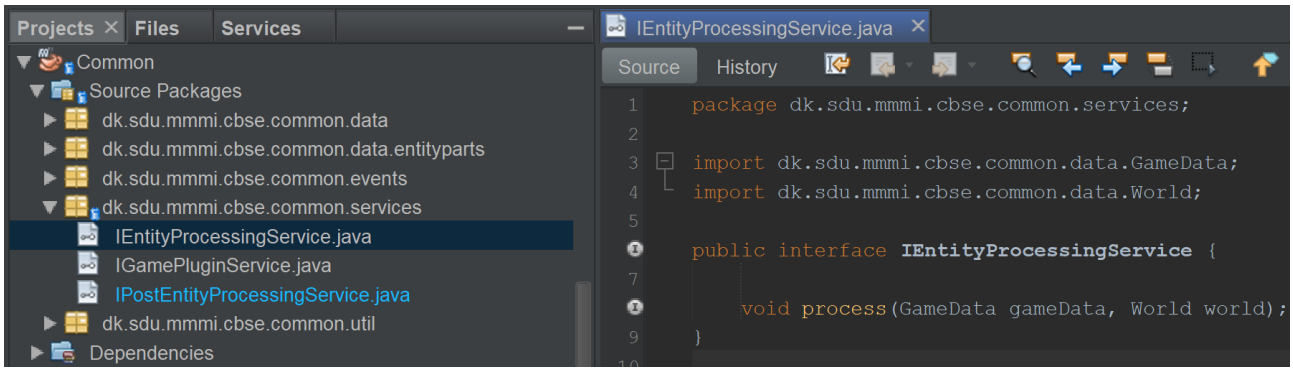
Anders Justesen
Anjus15@Student.sdu.dk

Javalab

Service provider interfacet defineres.



Her er tre interfaces defineret: IentityProcessingService, IGamePluginService og IPostEntityProcessingService.

Dernæst defineres den service som skal hente vores SPI'er.

Dette sker i klassen SPILocater.java, hvor den enkelte klasse mappes sammen med den enkelte ServiceLoader.

Hvad der så bliver mappet bliver hentet ind i Game.java med metoderne:

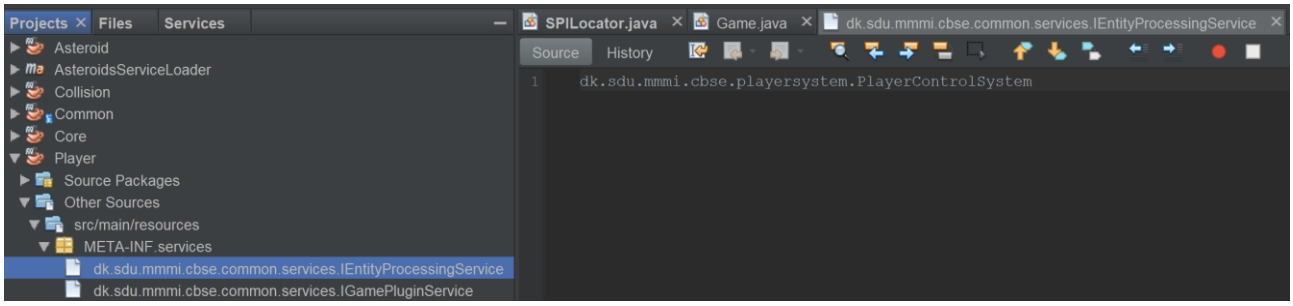
```
private Collection<? extends IGamePluginService> getPluginServices() {
    return SPILocator.locateAll(IGamePluginService.class);
}

private Collection<? extends IEntityProcessingService> getEntityProcessingServices() {
    return SPILocator.locateAll(IEntityProcessingService.class);
}

   private Collection<? extends IPostEntityProcessingService> getPostEntityProcessingServices() {
    return SPILocator.locateAll(IPostEntityProcessingService.class);
}
```

Nu skal vores Service provider registreres.

Dette gøres ved at oprette en mappe som hedder META-INF.services med en tom fil som indeholder det unikke navn på interfacet og navnet på implementeringen.

Her ses det at Player-modulet implementerer IEntityProcessingService og IgamePluginService:

Anders Justesen
Anjus15@Student.sdu.dk

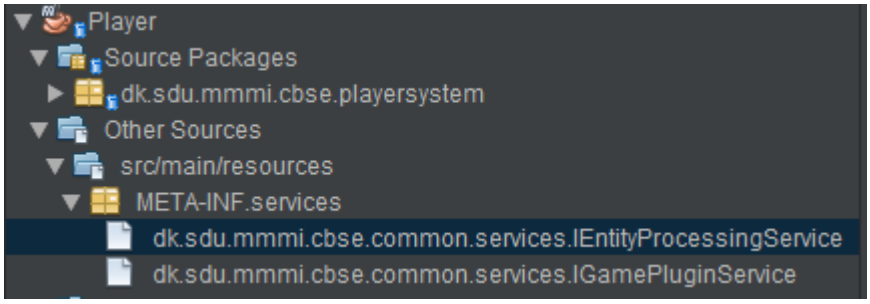


Så nu er Player en Entitet som kan tegnes via IentityProcessingService og kan bevæge sig ved at bruge IgamePluginService.

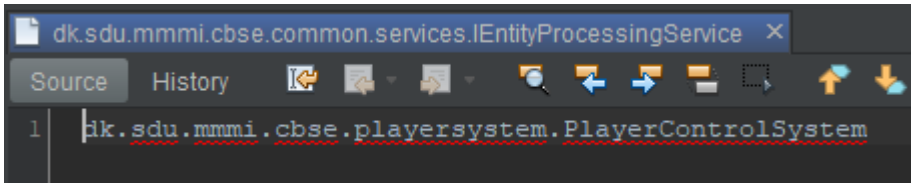
NetbeansLab1

Registrering af services.

Klassen og dens funktionalitet laves som fra forrige projekt. For at registrere at det er en service, oprettes en META-INF.services mappe, som indeholder vores registrering af servicen, her for player:



Inde i filen deklarerer man hvilke services der ”udbydes”:



Her deklarer man sin service som PlayerControlSystem, som bliver brugt igennem sit service provided interface som, i dette tilfælde, er IEntityProcessingSercive.

Inde i Core-klassen hvor spillet opdateres kaldes vores interfaces. Men de kaldes ikke som hardcoded interfaces, men benytter SPILocater-klassen som en hjælpeklasse.

```
private Collection<? extends IGamePluginService> getPluginServices() {
    return SPILocator.locateAll(IGamePluginService.class);
}
```

Den indeholder en metode som hedder locateAll, som bruger vores serviceloader. Serviceloaderen giver os et iterator baseret interface som vi kan iterere henover. Det bliver så tilføjet til en liste som returneres, som

Anders Justesen
Anjus15@Student.sdu.dk

vores update()-metode bruger i Game-klassen.

```
private void update() {
    // Update
    for (IEntityProcessingService entityProcessorService : getEntityProcessingServices()) {
        entityProcessorService.process(gameData, world);
    }
     for (IPostEntityProcessingService postEntityProcessorService : getPostEntityProcessingServices()) {
        postEntityProcessorService.process(gameData, world);
    }
}
```

NetbeansLab2

Inde i pom.xml filen under AsteroidsNetbeansModules-app vælges hvilke komponenter man vil have med over i sit update center.

```
<dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>Player</artifactId>
    <version>${project.version}</version>
</dependency>
<dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>Bullet</artifactId>
    <version>${project.version}</version>
</dependency>
```

F.eks. her vil Player og Bullet komponenterne blive en del af update centeret. Hvis jeg ikke ville have dem med, kunne de fjernes før update centeret oprettes.

Dernæst ændres konfigurationen for AsteroidsNetbeansModules-app fra ”<Default config>” til ”deployment”. Dette gøres for at man kommer ind og kører profilen med id ”deployment” og goalet ”autoupdate”. Nu ”Clean and Build” på AsteroidsNetbeansModules-app som så opretter mappen ”netbeans_site” i vores target mappe.

Nu har vi oprettet vores Update Center. Mappen ”netbeans_site” kan nu flyttes til en hvilken som helst plads. Her er det smart da dette kan flyttes et sted online og man vil derpå kunne foretage ændringer i sin applikation uden at slukke for applikationen. I vores tilfælde rykkes den til skrivebordet.

Dernæst skal vi have oplyst vores SilentUpdate modul om at Update centeret er flyttet. Dette gøres ved at gå ind i filen ”Bundle.properties” som ligger i /Other sources/src/main/ressources/org.netbeans.modules.autoupdate.silentupdate.ressources/.
Bundle-filen indeholder information omkring update centeret, deriblandt stien dertil. Den sti skal ændres så i mit tilfælde kommer stien til at hedde:
file:/C:/Brugere/Bruger/OneDrive/Skrivebord/netbeans_site/updates.xml

Projektet kan nu køres og modulerne kan loades og unloades dynamisk, som det fremgår af videoen.

Anders Justesen
Anjus15@Student.sdu.dk

OSGI Lab

BundleActivator.

Vi benytter BundleContext API'et sammen med BundleActivator interfacet til at oprette og registrere en kombonent. BundleActivator bruger start og stop metoder til at styre hvornår kombonenterne skal starte og stoppe. Dette gøres ved at man overskriver metoderne. Her er for Activator-klassen i OSGiLaser:

```
public class Activator implements BundleActivator {

    @Override
    public void start(BundleContext context) throws Exception {

        LaserSystem ls = new LaserSystem();
        context.registerService(IEntityProcessingService.class, ls, null);
        context.registerService(BulletSPI.class, ls, null);
    }

    @Override
    public void stop(BundleContext context) throws Exception {
    }

}
```

Her er BundleContext et parameter i start og stop metoden, som gør at vores "bundle" kan interagere med resten af frameworket, som gøres ved at registrere vores kombonent. Dette gøres ved at mappe den til et SPI.

For at maven-bundle plugin'et ved hvor vores BundleActivator er, oprettes der en osgi.bnd fil med indholdet: "Bundle-Activator: dk.sdu.mmmi.osgilaster.Activator". Når bundlet et bygget er der en .jar fil som vises i MANIFEST.INF-filen.

Declarative Service.

Når vi bruger declarative service, har vi ikke længere brug for en Activator. I stedet skal vi bruge en xml-fil som indeholder kombonentens beskrivelse.

Den xml-fil indeholder forskellige tags:

Implementation – specificerer lokationen af implementeringen af servicen.
Service-provide – Specificere det SPI, som kombonenten mapper til i service registeret.
Reference – Bruges til at bruge andre kombonenter, som kan bruges ved bind- og unbind-metoder.