Tællende Aktivitet 1

SB4-KBS

Anders Lyngholt Højgaard Larsen

Larse19@student.sdu.dk

Kildekode

Kildekoden kan findes på github, hvor videoerne også kan findes under deres tilhørende lab.
https://github.com/larse19/SB4-KBS-CA1

JavaLab

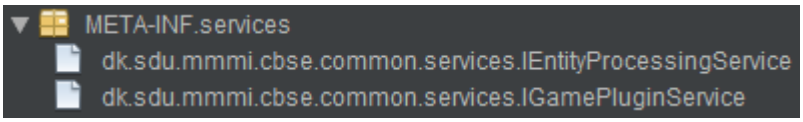
I dette lab bliver en service lokaliseret vha. Javas ServiceLoader. I Common modulet ligger klassen, SPILocater, som håndterer dette. Denne klasse har en statisk metoder, som hedder "locateAll()". Denne metoder tager en vilkårlig klasse som argument, og bruger ServiceLoaderens indbyggede metode, "load()", til at oprette et ServiceLoader instans, for den type. Der bliver holdt styr på disse serviceloaders i et hashmap, med servicen som key. Dette sørger for at der kun bliver oprettet én serviceLoader for hver service, da dette map bliver tjekket, før en ny bliver oprettet.
Der bliver da itereret igennem denne serviceLoader, og hver instans af typen, bliver tilføjet til en list<T>, som bliver returneret af locateAll() functionen.

I Game bliver denne metode da brugt til at loade alle providers. De providers der implementerer IGamePlugin interfacet, bliver loaded i create(), hvor der bliver itereret igennem dem, og start() funktionen bliver kaldt på dem alle sammen.

Providers af typen IEntityProcessingService og IPostEntityProcessingService, bliver loaded i update funktionen, hvor de også bliver itereret igennem, og process() bliver kaldt på hver af dem.

Alle services er beskrevet i common modulet, i form af interfaces. Så for at tilføje en provider, skal man tilføje et modul med mindst én klasse der implementerer en af disse services.

Hvert modul har en META-INF.services mappe, som indeholder filer der beskriver stien, til den klasse der implementerer en service. f.eks:



Her indeholder hver af disse filer, pakkestien til implementationen af hver af disse services, som f.eks her med implementationen af IEntityProcessingService, i Player modulet:

```
dk.sdu.mmmi.cbse.playersystem.PlayerControlSystem
```

Service locatoren bruger disse filer til at finde frem til implementationerne.

NetbeansLab 1

Jeg brugte det givne eksempel, NBRuntimeExample12, som grundsten for netbeans aplikationen. Her kunne man dog simpelt havde oprettet et nyt, da eksemplet indeholder meget lidt ekstra tilføjet ting.

Jeg oprettede da et et nyt NetBeans modul for hvert modul i spillet; Common, Core, Player, Asteroid, Collision og Bullet. Selve spillogikken kunne nemt overføres fra tidligere lab (JavaLab), ved at kopire pakkerne ind i modulet. For at NetBeans kunne genkende de forskellige services, tilføjede jeg @ServiceProvider keywordet til toppen af hver klasse, som implementerede en service. Dette gør at netbeans automatisk kan oprette en MATA-INF.services mappe (som nævt i tidligere lab).

For at starte spillet, tilføjede jeg en Install.java fil til core. Denne fil bliver kørt når modulet bliver installeret, og indeholder en restore() metode, som bliver kaldt ved installeringen. Denne metode inderholder den funktionalitet, som skal køre ved installering. Så den funktionalitet der før lå i main.java, er nu flyttet ind i denne metode, og main.java er blevet slettet.

Hvor man før søgte efter provider med SPILocater klassen, bruger man nu NetBeans lookup api. Denne api kan gennemsøge projektet og finde alle providers af en bestemt klasse. Denne API gør brug af whiteboard modellen, som betyder at den hele tiden stå og holder øje med providerne, og hver gang en af dem ændrer sig, så “sender” den en opdatering ud til dem der lytter efter ændringerne. Dette kaldes også for et observer pattern.

NetBeansLab 2

I dette lab bliver der tilføjet et update center til projectet, som gør at man kan loade og unloade moduler i runtime. Dette er gjort ved at sætte appens konfiguartion til “deployment”, hvilket opretter en netbeans_site mappe, som indeholder en updates.xml fil. Denne fil indeholder alle de moduler loadede moduler i projektet. Denne mappe kan man da klippe ud af projektfolderen, og gemme den et andet sted, jeg lagde den på skrivebordet. Derefter tilføjes et modul der hedder SilentUpdate, som er blevet udleveret. Dette modul bruger netbeans autoUpdate api, til at holde øje med opdateringer i update.xml filen, og installer og uninstaller de moduler som bliver tilføjet og fjernet fra filen. Stien til update.xml er defineret i SilentUpdate modulet bundle.properties fil. Appen bliver da konfiguret til at være default igen, og når man nu kører appen, kan man unloade moduler ved at fjerne den fra xml filen (Dette kan ses i den vedlagte video), og ved at gemme filen, forsvinder det modul automatisk fra appen, i runtime. Det samme kan gøres når man skal tilføje et modul, her skriver man den bare ind i xml filen, og det bliver automatisk tilføjet til appen.

OSGILab

Her har jeg bygget videre på det udleverede projekt, PaxAstroids. Jeg har da tilføjet en player ved at oprette et nyt modul, og tilføjet funktionaliteten fra tidligere projekt til to java klasser, PlayerPlugin og PlayerProcessor (sidstnævnte indeholder funktionalitet fra PlayerControlSystem fra tidligere projekt). Derudover er der oprettet en klasser, der hedder Actiavtor, som implementerer BundleActivator fra OSGi framworket. Denne klasse aktiverer de services, som modulet provider vha. BundleContext klassen. Denne klasse indeholder en metode som hedder; registerService(), som instantiere en implementation af en service, og beskriver hvilken service det er en provider af.

Projektet bliver kørt med pax provider, som giver en CLI hvor man kan stoppe og starte specifikke moduler, som også kan ses i videoen.