Technical Design

"To optimise every gymnast training-goals in a fun, safe and efficient way"

Preface

a) Intro
b) UML Class Diagram: 4e versie
c) Activity Diagram COMPLETE & STUCK
d) Architectuur
e) Assumptions and Considerations

Update #3

- Class Diagram aangepast zonder u.i.
- Architectuur toegevoegd met de modulaire opbouw uit leeruitkomst.
- Considerations aangevuld.

Update #4

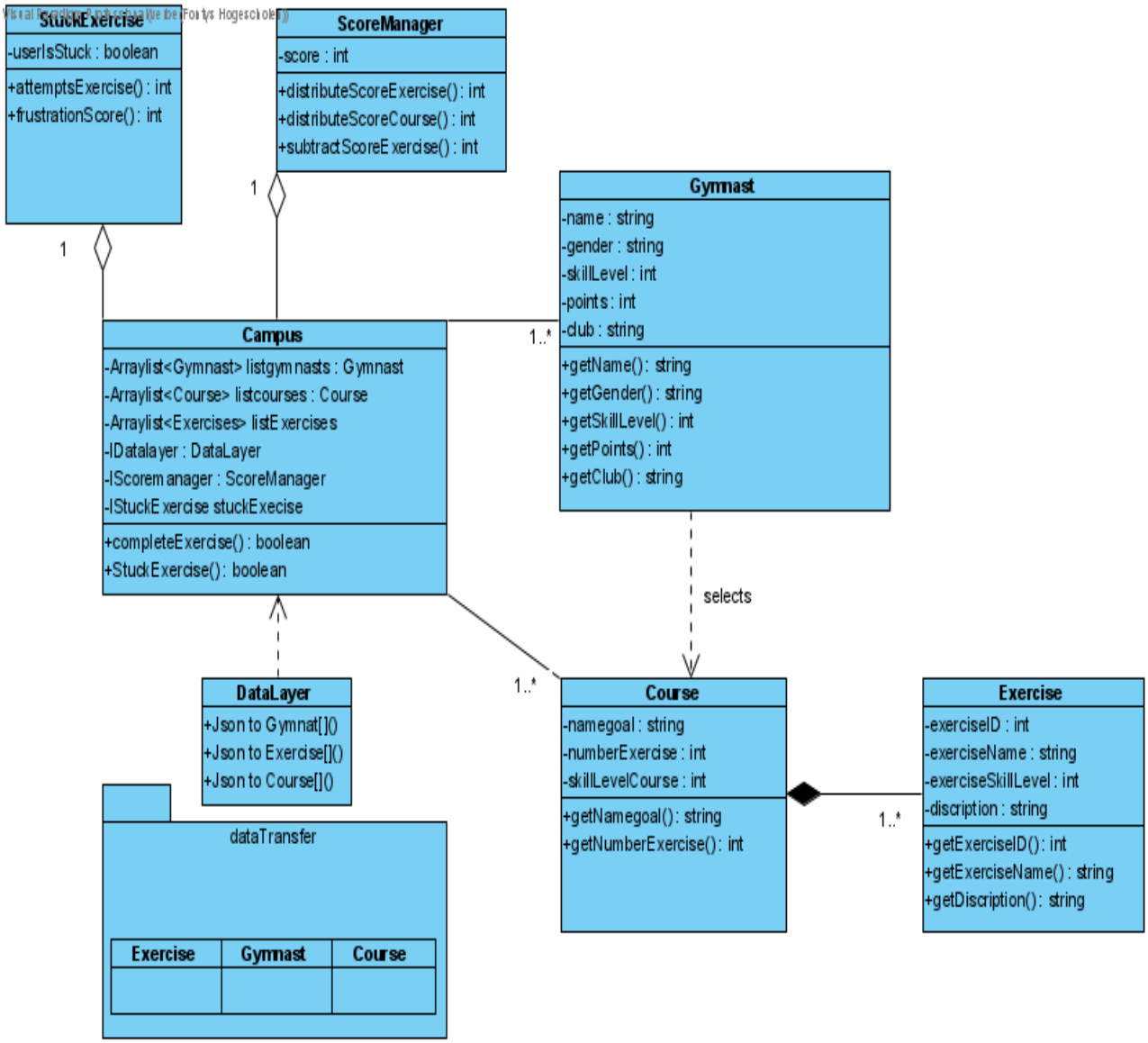
- Class Diagram aangepast met juiste return methode.
- Activity diagram STUCK toegevoegd
- Architectuur verder toegelicht en beschreven hoe ik de datalaag injecteer in de logicalaag.
- Considerations aangevuld m.b.t. interfaces en modulaire opbouw.

Intro

In dit technische design beschrijf ik de belangrijkste opbouw van mijn applicatie en komen bepaalde keuzes in ontwerp aan het licht. Hoewel ik gedurende het semester heb geworsteld met het ontwerp van mijn klassendiagram, begrijp ik steeds beter hoe ik tot de kern moet komen. Door het beter begrijpen van de theorie heb ik ook bepaalde beslissingen aangepast en toegevoegd. Dit hele proces zorgt ervoor dat je op het einde goed kunt terugzien wat er allemaal geleerd is in 1 semester. Zo ook dit technische ontwerp.

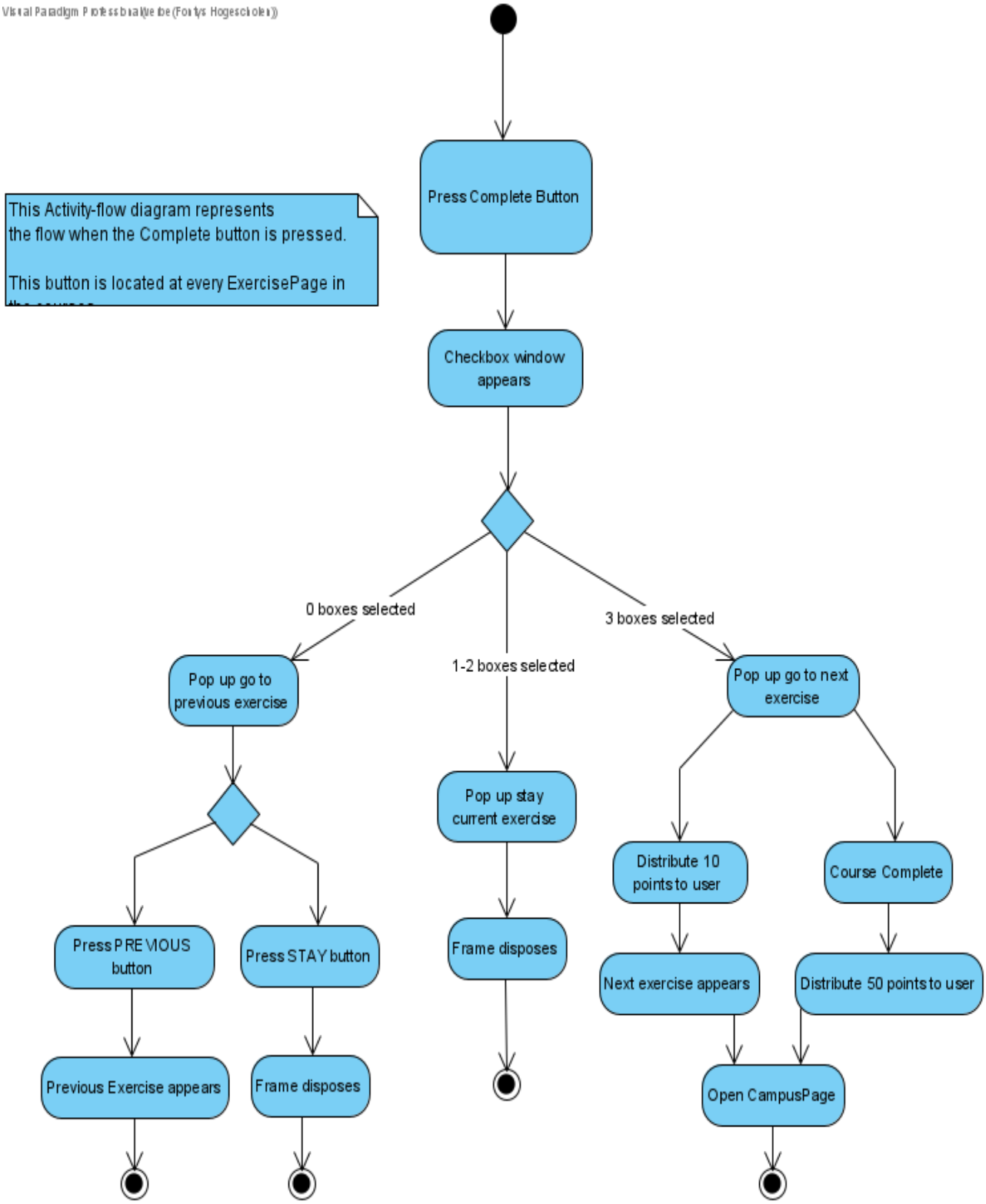
Class Diagram #4: Zonder Ui

Na een tip van John om de UI zoveel mogelijk uit het klassendiagram te halen heb ik hier de logica-laag in een klassendiagram geplaatst. Ook heb ik methodes die eerst leeg waren nu het juiste return-type gegeven.

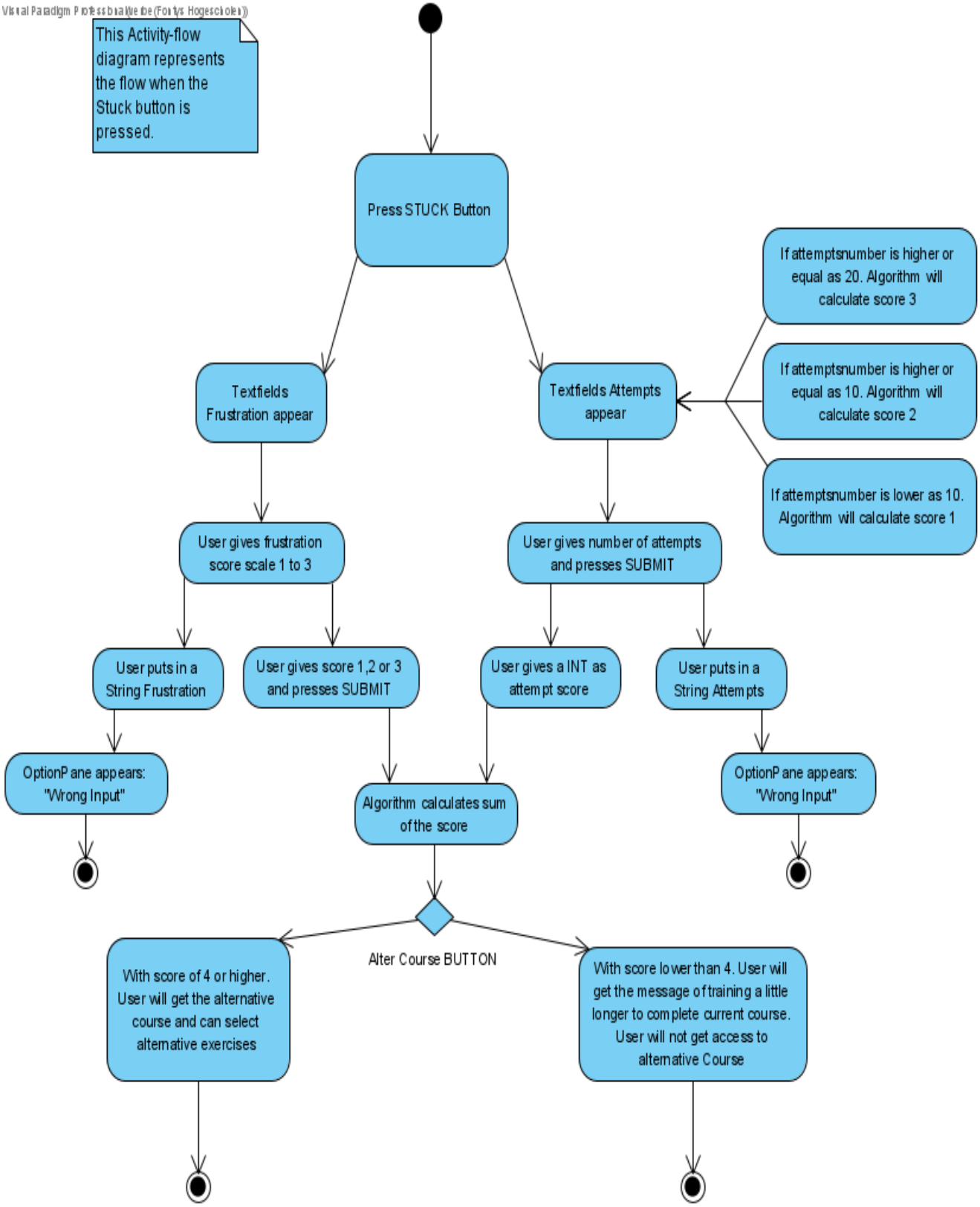


Activity Diagram

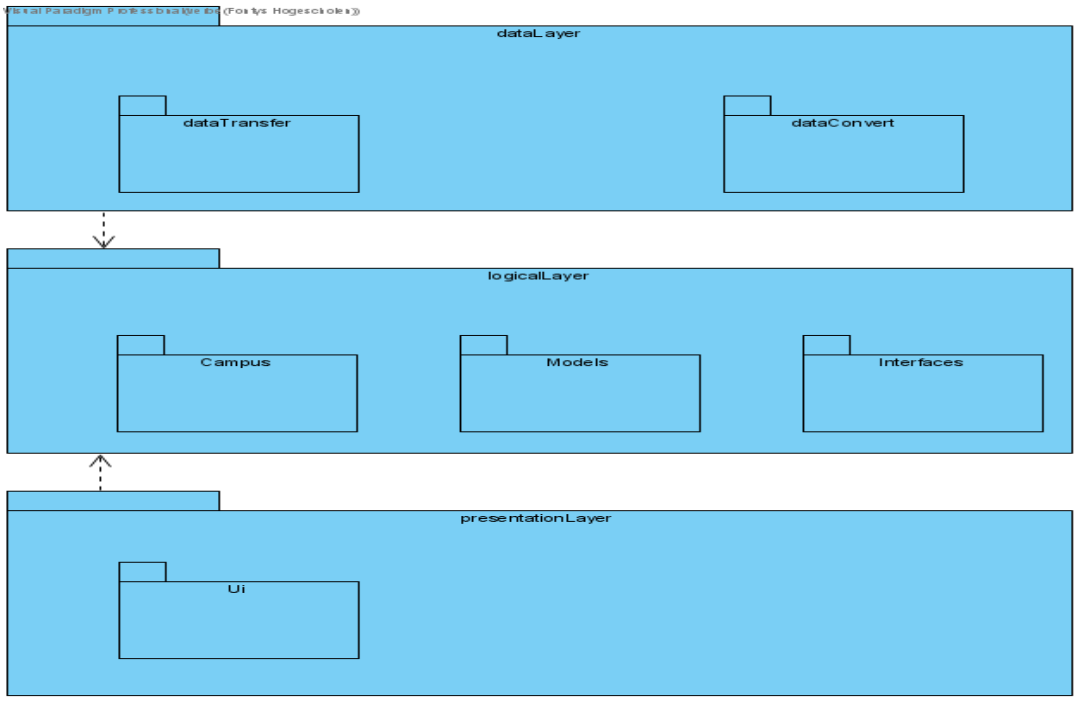
Activity Diagram NEXT



Activity Diagram STUCK



Architectuur



In de eerste versies van mijn implementatie had ik mijn code opgebouwd zonder goed na te denken over de uitbreidbaarheid van de applicatie. Ik gebruikte logica, data en ui door elkaar en kwam erachter dat ik veel te veel verantwoordelijkheden had staan bij een aantal specifieke klassen.

Allereerst ben ik dit gaan aanpakken door te werken met packages. Hier probeerde ik al een onderscheid te maken tussen de verschillende lagen uit mijn applicatie. Ook ging ik gelijktijdig aan de slag met de verantwoordelijkheden per klas. Ik probeerde deze steeds meer en meer op de juiste plekken te krijgen en eventueel onder te brengen in nieuwe klassen.

Dit resulteerde in een veel logischere opbouw waar het voor iedere lezer duidelijk is waar de verschillende functies van de app terugkomen. Na de instructie over N-tier-architectuur snapte ik goed het belang van deze lagen en wilde ik deze module-opbouw ook terug laten komen in mijn eigen ontwerp. Dit lukte me niet in mijn huidig aangemaakte project. Het is me echter wel gelukt in een nieuw aangemaakt project. Hierin heb ik de lagen aangemaakt en de code verplaatst naar de juiste plek.

- dataLayer
- logicalLayer
- presentationLayer

Door deze stap is het gebruik van een interface op dit vlak duidelijk geworden en snap ik hoe dit ervoor zorgt dat de logica-laag los van de data getest kan worden met mock-data. Dit naast de kansen om de modules met verschillende systemen en bronnen te laten werken.

Assumptions & Considerations

1. Java constructie: JSwing
   In mijn applicatie heb ik gekozen voor een “Java Swing” build. De belangrijkste reden voor deze keuze is de visualisatie van de verschillende concepten voor mijzelf. Bij de start van de bouw hielpen het frame, de verschillende buttons en de labels mij om te begrijpen waar ieder stukje van de code thuishoort. Ook viel het mij op dat het me ook hielp met het structureren van mijn project op meerdere manieren
   - Scheiden van UI, Object Classes en de Data.
   - Het opbouwen van Parent en Child-classes voor de uitbreidbaarheid van de code door het visualiseren van echte schermen.
   - Het leren van een nieuwe codetaal in een nieuwe codetaal.
     Ondanks dat ik een beginner ben vond ik het toch belangrijk om dit semester kennis op te doen van het samenstellen van een GUI. Voor mijn gevoel is dit toch een onderdeel dat een applicatie compleet maakt, ongeacht dit niet direct een eis in de leeruitkomsten is. Ik heb daarom wat extra uren gereserveerd om deze vaardigheid eigen te maken.

2. Van Datadump naar Json
   Na het zien van de Json structuur bij een project van John heb ik deze ook toegevoegd in mijn project. De leesbaarheid is overzichtelijker en niet onbelangrijk is JSON nagenoeg platform-onafhankelijk. Dit laatste is de belangrijkste reden voor mijn keuze. Om uitbreidbaarheid te stimuleren is data één van de belangrijkste onderdelen. Daarom leek me de veelal gebruikte JSON structuur een prima keuze. Ook omdat deze goed zichtbaar is in het project zelf waardoor ik het overzicht goed kan houden. Het duurde even om de juiste jars te downloaden (en uberhaupt te leren over Maven en Jars), maar uiteindelijk is het met de juiste syntax gelukt. Op dit moment converteer ik mijn modellen naar JSON files.

3. Interfaces v.s. Inheritance
   Het eerste gedeelte van dit semester had ik mijn project opgebouwd m.b.v. Inheritance. Mijn G.U.I was opgebouwd uit allerlei nieuwe pagina’s die elkaars logica overnamen. Inmiddels ben ik helemaal overgestapt op een ander ontwerp. Ik gebruik een interface om de datalaag te injecteren in mijn “Mission Control” om deze los te koppelen. Ook gebruik ik de inheritance niet meer in de GUI en gebruik ik nu een overzichtelijke pagina die met methodes veranderd i.p.v. constant nieuwe pagina’s aanmaakt en hierdoor data vergeet/verliest.

4. Architectuur: N-tier
   N.a.v. een instructie over modulaire opbouw wilde ik dit snel en graag in mijn applicatie toepassen. Het idee van het loskoppelen van data, logica en presentatie sprak me aan. John adivseerde mij te starten met de opbouw in packages. Dit kreeg gek genoeg veel foutmeldingen in mijn oude project. Daarom heb ik ervoor gekozen om een nieuw project te starten en een handleiding te volgen om te starten met een juiste modulaire opbouw. Dit lukte sneller dan ik dacht en kreeg ik werkend binnen een aantal uur. Het enige wat nog werk vereist is de juiste dependencys neerzetten van de verschillende modules (nu nog circulair).