Object Oriented Analyses en Design

Clover Golf



**Klas: IS201**

**Naam: Mohamed, Alwin**

Inhoud

[Casus. 3](#_Toc401568498)

[Fully-Dressed Use Cases 4](#_Toc401568499)

[Use Case Diagram 6](#_Toc401568500)

[Model Diagram 7](#_Toc401568501)

[Sequence Diagram 8](#_Toc401568502)

[Contracts 12](#_Toc401568503)

[Deployment Diagram 13](#_Toc401568504)

[Design class diagram 14](#_Toc401568505)

[Design sequence diagram 15](#_Toc401568506)

# Casus.

Golf Applicatie en web winkel promotie:

Een Golf vereniging wil een mobile applicatie ontwikkelen voor de Golf community.

Met de applicatie wil de Golf vereniging alle Golf wedstrijden, spelers, statistieken, tijden en uitslagen en nog veel meer beschikbaar maken voor de Golf lief hebbers over de hele wereld.

Naast de free applicatie kan men zijn account upgraden naar een member status, met als voorwaarde dat de persoon lid is van de golf vereniging.

Member leden krijgen exclusieve service zoals, velden huren, contributie via de applicatie betalen, korting krijgen op assortimenten, Golf profiel aanmaken, Golf trainer huren.

**Doel:**

-Golf community up-to-date houden over Golf statistieken.

-Als financiële doeleinde wil de Golf vereniging het web winkel voor high premium sport uitrustingen

promoten.

-Royale klanten werven.

**Member**:

Betalen contributie, Fully-Dressed

Huren golfveld,

Maken profiel,

Toon korting bon

Beheer account,

Huren personeel assistent.

**Visitor**:

Opvragen statistieken, Fully-Dressed

Opvragen Evenementen

Bestellen tickets,

Bekijk agenda

Koppelen Twitter/Facebook,

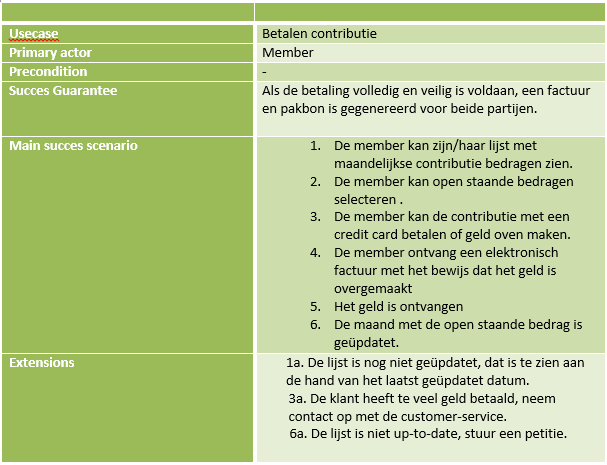
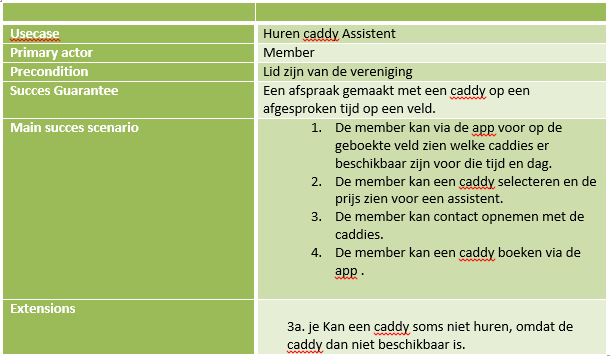
Kopen artikelen, Fully-Dressed

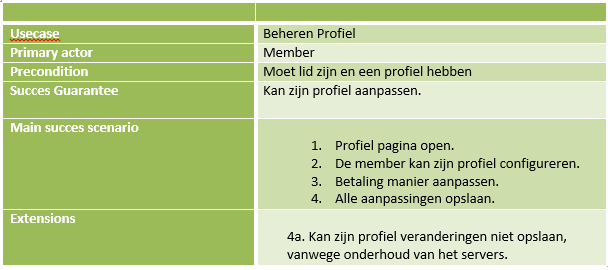
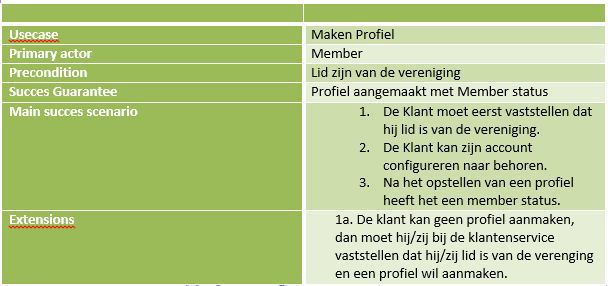
huren Trainer.

**System-Administrator**:

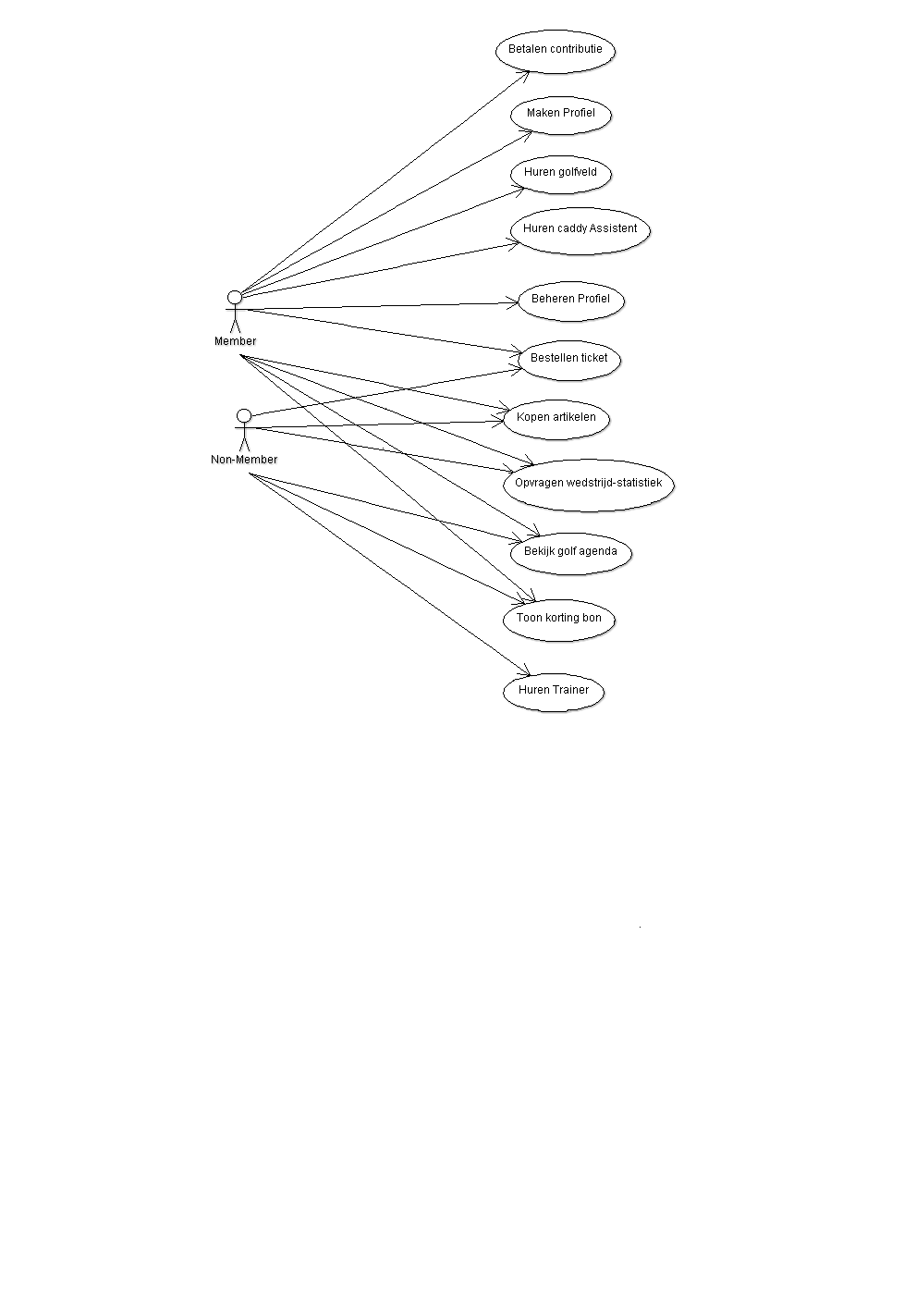
Beheren applicatie.

# Fully-Dressed Use Cases



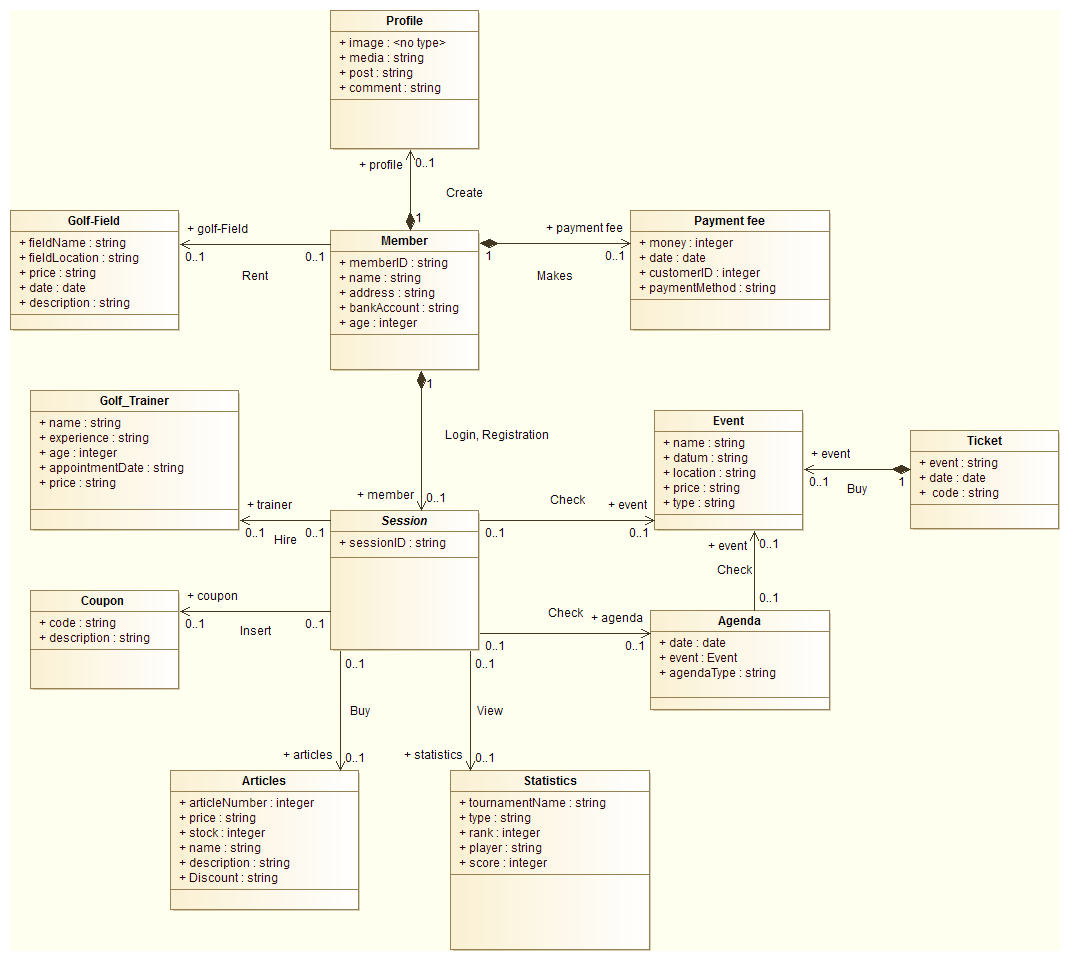


# Use Case Diagram



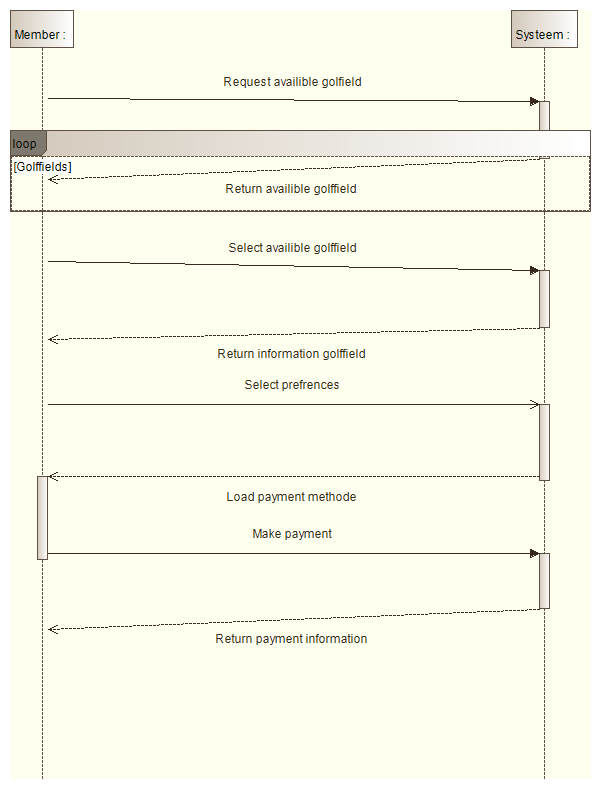
Dit afbeelding figuur 1. laat een user case diagram zien van Clover Golf

# Model Diagram

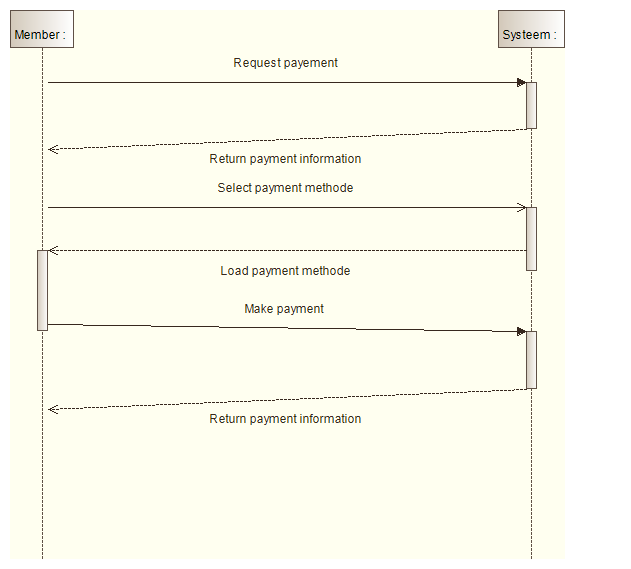


Dit afbeelding figuur 2 laat een model diagram zien van Clover Golf

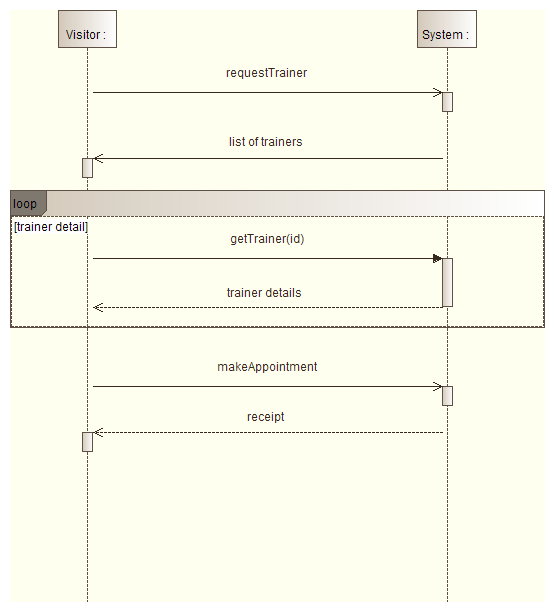
# Sequence Diagram



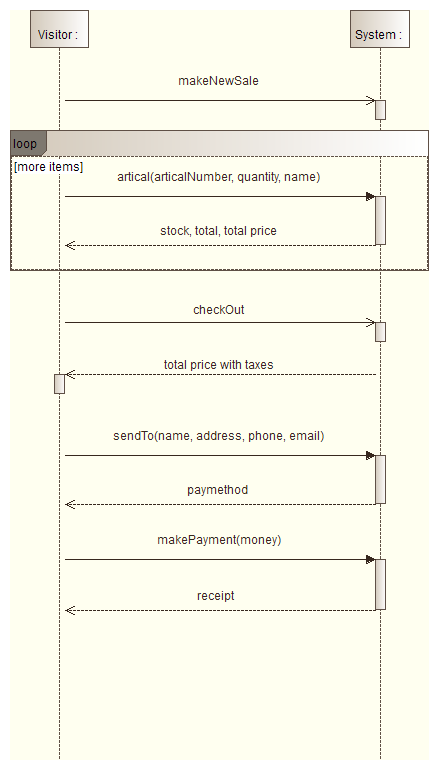
Dit afbeelding figuur 3 laat een sequence diagram zien van de use case huren golfveld



Dit afbeelding figuur 4 laat een sequence diagram zien van de use case huren golfveld



Dit afbeelding figuur 5 laat een sequence diagram zien van de use case huren trainer



Dit afbeelding figuur 6 laat een sequence diagram zien van de use case betalen contributie

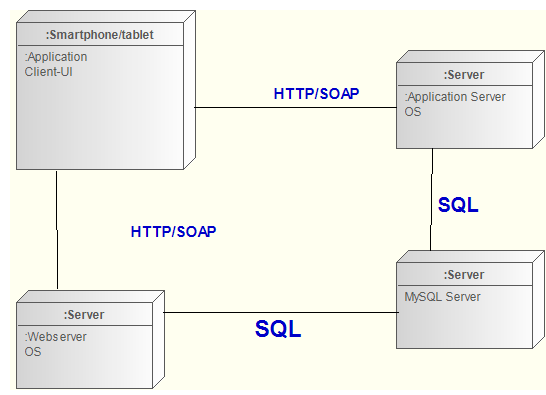
# Contracts

|  |
| --- |
| Operation Contracts:  Operation : createOrder()  Cross References:Use Cases: kopen articles.  Preconditions: klant is aan het winkelen in de web shop.  Postconditions: -klant maakt een order  -een artikel wordt opgevraagd met de aantal.  -de artikel en de aantal wordt aan de lijst toegevoegd.  -de artikel wordt gezocht in de database.  - een artikel instance article wordt gecreëerd(instance creation)  - artikel.articalNumber, quentity en name worden artikel(attribute modification)  -de artikel wordt terug gegeven  -De systeem kijkt of er nog een artikel wordt toegevoegd. Zo ja, dan begint het vanaf stap 2 opnieuw  -De systeem geeft de totaal bedrag terug.  -De klant kan de betalen maken.  -De klant krijgt een bon terug. |

|  |
| --- |
| Oparation Contracts:  Operation: makePayment()  Cross References: UseCase: Betalen contributie  Precondition: Member request payment  Postcondition:   * Een payment instantie is gecreëerd(instantie creatie) * De totaal bedrag wordt gecalculeerd en gecontroleerd * De betaal informatie wordt verzonden naar de gebruiker. |

|  |
| --- |
| Oparation Contracts  Operation: getGolfFields()  Cross References: UseCase: Huren golfvelden  Precondition: Ingelogd als member  Postcondition:   * De member maakt een instantie aan van golffield * De golffield controller maakt een lijst van alle golfvelden. * De golffield controller geeft alle golfvelden terug binnen een loop. * De golffield controller laat alle beschikbare golffields zien |

# Deployment Diagram

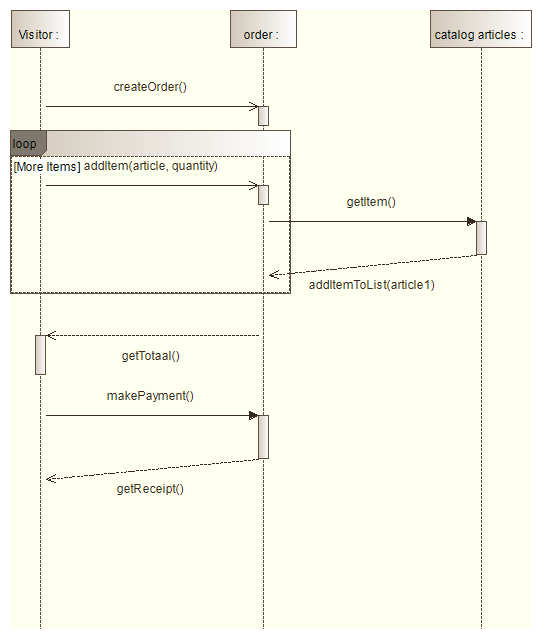


Dit afbeelding figuur 7 laat een deployment diagram zien van Clover Golf

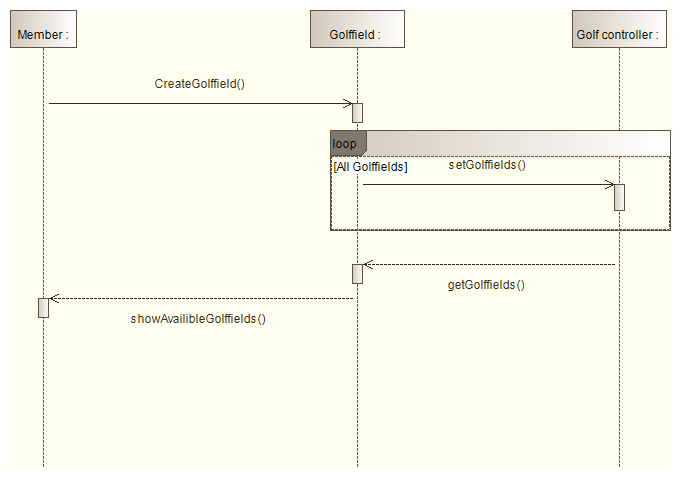
# C:\Users\Hokaki\Documents\GitHub\ULM-Golf\ooad Class diagram.pngDesign class diagram

Dit afbeelding figuur 8 laat een design class diagram zien van Clover Golf

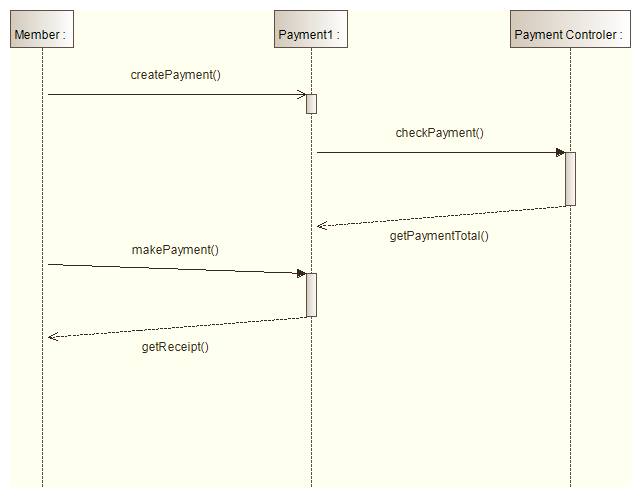
# Design sequence diagram



Dit afbeelding figuur 8 laat een design sequence diagram zien van de contract createOrder()



Dit afbeelding figuur 9 laat een design sequence diagram zien van de contract getGolfFields ()



Dit afbeelding figuur 10 laat een design sequence diagram zien van de contract makePayment()

# Grasp

Deze grasp is voor design sequence diagram figuur 8 createOrder().

Name: Creator

Problem: Klant wilt een order maken.

Solution: Klant maak een instantie Order, zodat Order een instantie van Itemlist kan maken. Itemlist kan items toevoegen.

Name: Information expert

Problem: The klant wilt artikelen bestellen. Daarvoor heeft de klant een order nodig.

Solution: De order heeft de methods om artikelen toe te voegen aan de instantie.

Name: Low Coupling

Problem: Hoe kan ik ervoor zorgen dat objecten minder afhankelijk zijn van elkaar.

Solution: Door niet te veel referenties te maken naar het object. Ik kies ervoor om het object meer centraal te maken.

Name: Controller

Problem: Wat is de eerste object dat de UI ontvangt?

Solution: De eerste object is de itemlist. De klant kan daar items aan toe voegen en verwijderen.

Name: High Cohesion

Problem: Hoe kan ik objecten simpel en beheersbaar maken?

Solution: Door de objecten simple te maken met weinig methods.

Deze GRASP is voor Design Class Diagram figuur 8. sequence of assignment getAgenda(), getEvent(), requestTicket(), buyTicket.

Name: Creator

Problem: De klant wilt een ticket kunnen bestellen.

Solution: De klant maakt een instantie van Agenda en event en ticket. Waardoor de klant de actie kan uitvoeren om een ticket te kopen.

Name: Information Expert

Problem: hoe kunnen we weten welke evenementen er beschikbaar zijn voor de klanten.

Solution(advice): Doordat de klant via de agenda de getAgenda() methode aanroept. Krijgt hij een UI terug met de agenda. Hierop staan alle events ingeladen.

Name: Low Coupling

Problem: Wat heeft de minimaal nodig om een ticket te kopen?

Solution(advice): doordat je via de agenda de datum selecteert wordt de methode getEvent aangeroepen. Vervolgens kan hij requestTicket() gebruiken om buyTicket() aan te kunnen roepen.

Name: Controller

Problem: Wat is het eerste object dat de UI event krijgt?

Solution: De klant krijgt als eerste de agenda te zien met de events.

(advice)

Name: High Cohesion

Problem: Hoe kan je ervoor zorgen dat de object van agenda beheersbaar is.

Solution: door duidelijke referenties tussen agenda en event te maken.

(advice)

Deze GRASP is voor Design Class Diagram figuur 8. sequence of assignment voor het huren van een Golf\_Trainer.

Name: Creator

Problem: De klant wil een trainer huren.

Solution(advice): De klant maakt een instantie van Golf\_Trainer. De klant kan de actie uitvoeren om een trainer te huren op de gewenste datum en tijd.

Name: Information Expert

Problem: Hoe kunnen de klanten zien welke trainers beschikbaar zijn voor de klant.

Solution(advice): Door de klant via getGolfTrainer een lijst van alle trainer krijgt en dan de gewenste trainer kan selecteren.

Name: Low Coupling

Problem: wat heb je minimaal nodig om accuraat een golf trainer te huren.

Solution(advice): Je hebt nodig setAppointmentDate() en setName().

Name: Controller

Problem: wat is het eerste object dat de UI Golf\_Trainer binnen krijgt

Solution(advies): Via session HireTrainer(String name, Date appointmentDate) wordt deze informatie naar Golf\_Trainer gestuurd.

Name: High Cohesion

Problem: Hoe kan je ervoor zorgen dat de object van Golf\_Trainer beheersbaar blijft?

Solution(advies): door dat de appointmentDate en name goed overeenkomt met setters van Class Gold\_Trainer.