Software Design Description

OOSE OOAD casus - Quebble

Sjoerd Scheffer
579392

HAN_UNIVERSITY
OF APPLIED SCIENCES

Docent: Marco Engelbart

10 juni 2021

Versie 1

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

1 Inleiding

1.1 Algemene beschrijving

Zie de Software Requirements Specification.

1.2 Doel van dit document

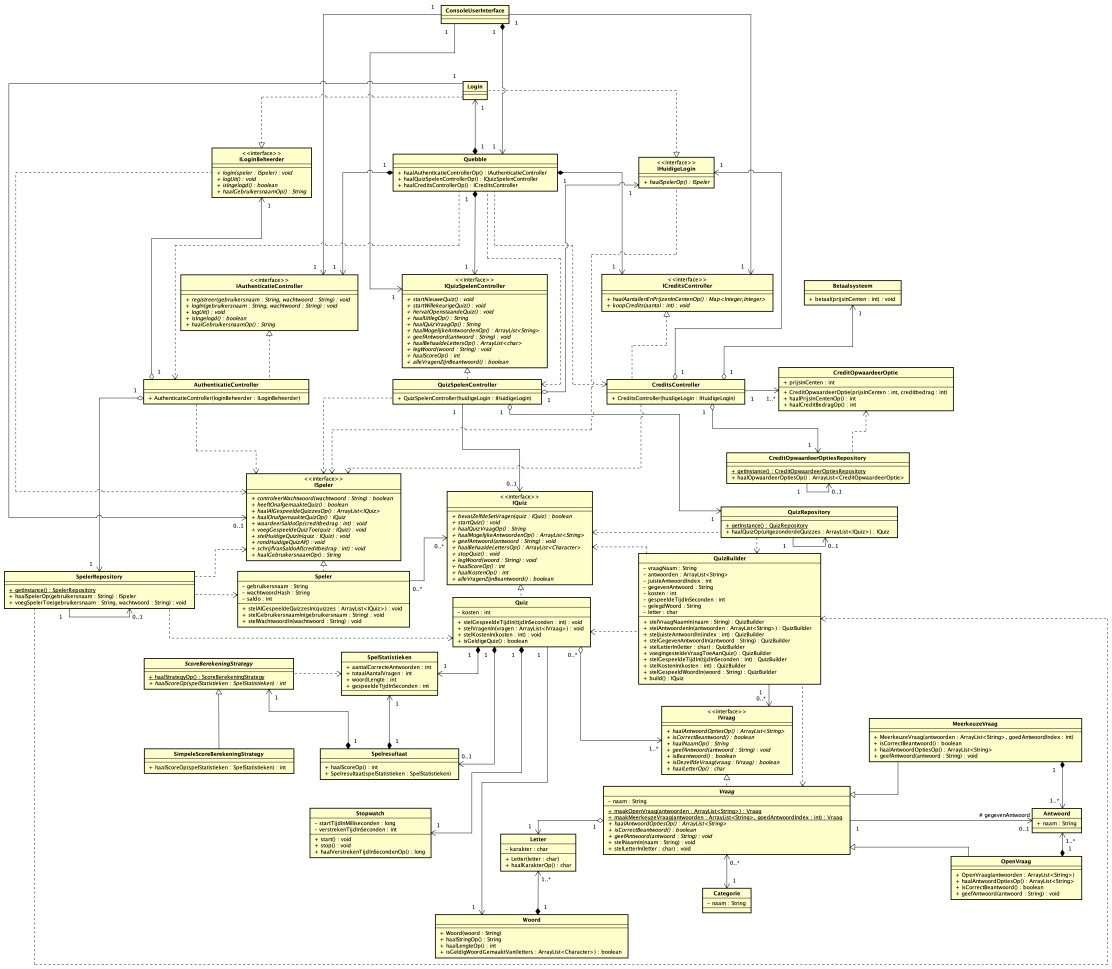
Dit document beschrijft de technische vertaling van de uitgewerkte usecases uit het Software Requirements Specification. Dit document beschrijft het ontwerp van Quebble als opgesteld aan de hand van de casusopdracht.

2 Gedetailleerde ontwerpbeschrijving

2.1 Ontwerp subsysteem Quebble

2.1.1 Klassendiagram

Deze sectie bevat het klassendiagram. Deze is gebaseerd op het domeinmodel uit de Software Requirements Specification. Daarop volgt een toelichting voor de afwijkingen van het domeinmodel.



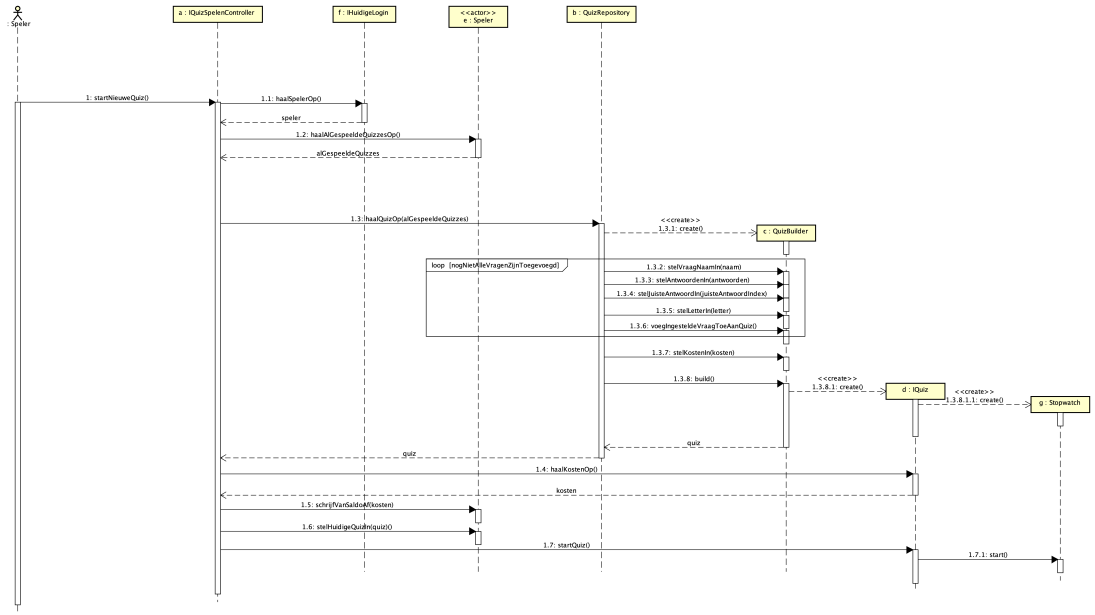
Figuur 1: Klassendiagram

De gebruikte patterns en principes uit tabel 1, 2 en 3 van sectie 2.1.3 zorgen voor verschillen met het domeinmodel uit de Software Rquirements Specification. Daarnaast zijn er nog een aantal verschillen:

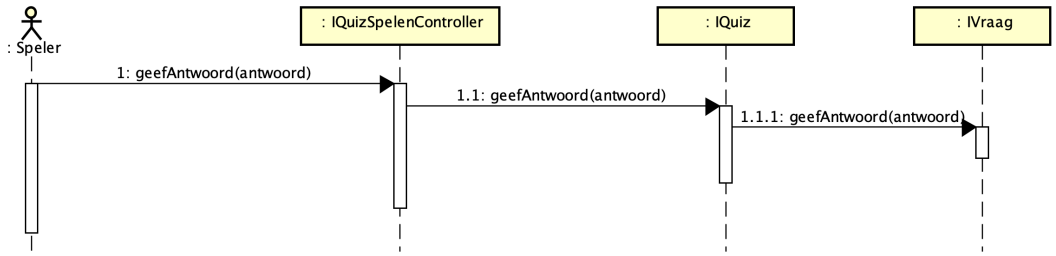
- De associatie tussen aan de ene kant **Speler** en aan de andere kant **Antwoord**, **Vraag** en **Woord**. Dit ten behoeve van het GRASP-principe Low Coupling en het daarbij toegepaste pattern Indirection (Larman, 2002).
- Het attribuut *goed* van **Antwoord**. Dit attribuut bestaat niet in het klassendiagram omdat de verantwoordelijkheid voor de goedkeuring bij de **Vraag** ligt. Anders zouden er bijvoorbeeld meerdere goede antwoorden bij een meerkeuzevraag kunnen zitten.
- De klasse **Creditbedrag**. Dit is een refactoring omdat het object geen verantwoordelijkheid had naast het bijhouden van een enkel aantal credits (Refactoring.Guru, g.d.). De klasse veroorzaakte alleen maar rommeligheid in het ontwerp.
- Alle klassen in het klassendiagram die geen dependency/associatie zijn van QuizController. Het domeinmodel beschrijft alleen de concepten voor het spelen van een quiz.

2.1.2 Sequencediagrammen

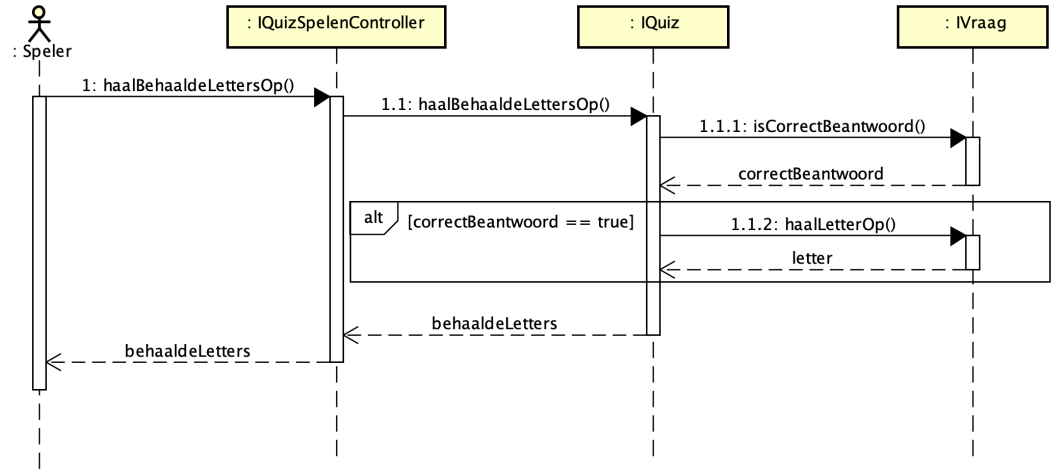
Deze sectie bevat sequence diagrammen die de belangrijkste systeemoperaties laten zien van de uitgewerkte usecases van de Software Requirements Specification.



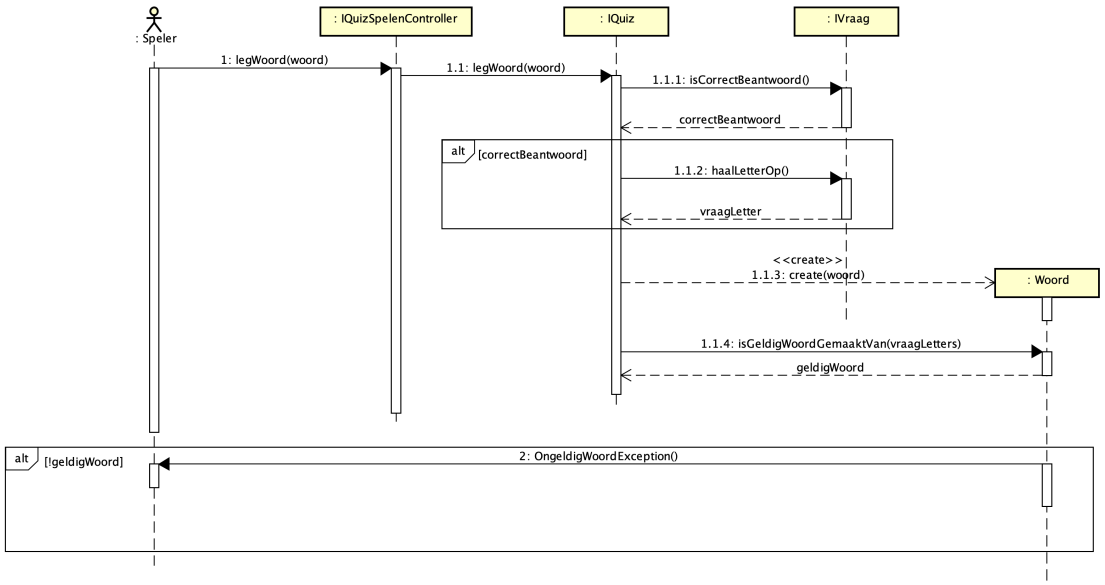
Figuur 2: Sequencediagram voor systeemoperatie *startNieuweQuiz*



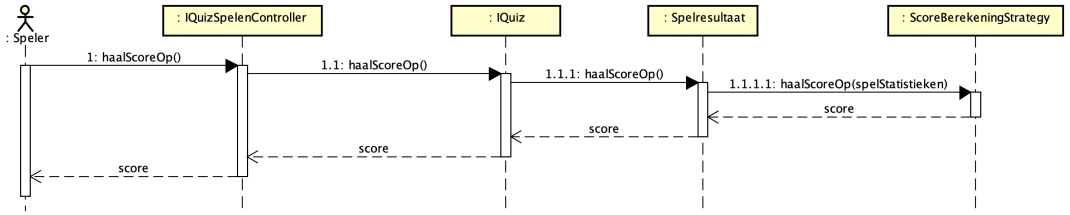
Figuur 3: Sequencediagram voor systeemoperatie *geefAntwoord*



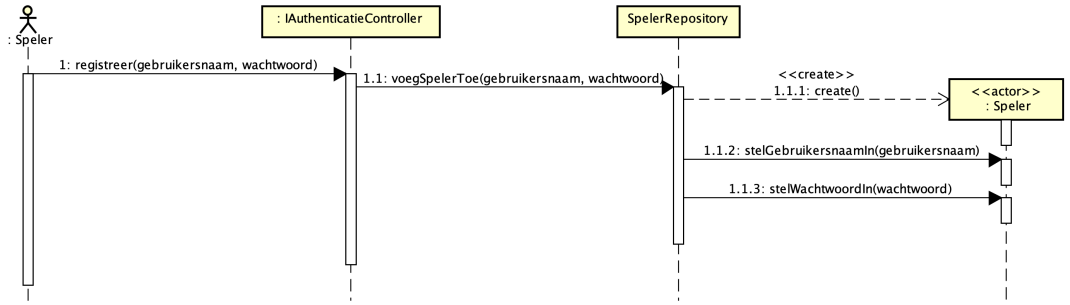
Figuur 4: Sequencediagram voor systeemoperatie *haalBehaaldeLettersOp*



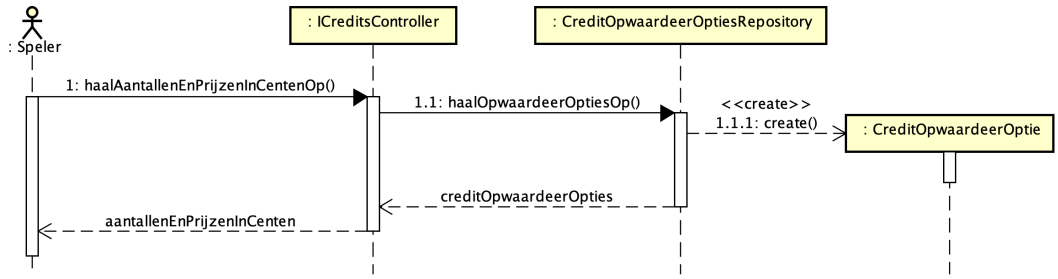
Figuur 5: Sequencediagram voor systeemoperatie *legWoord*



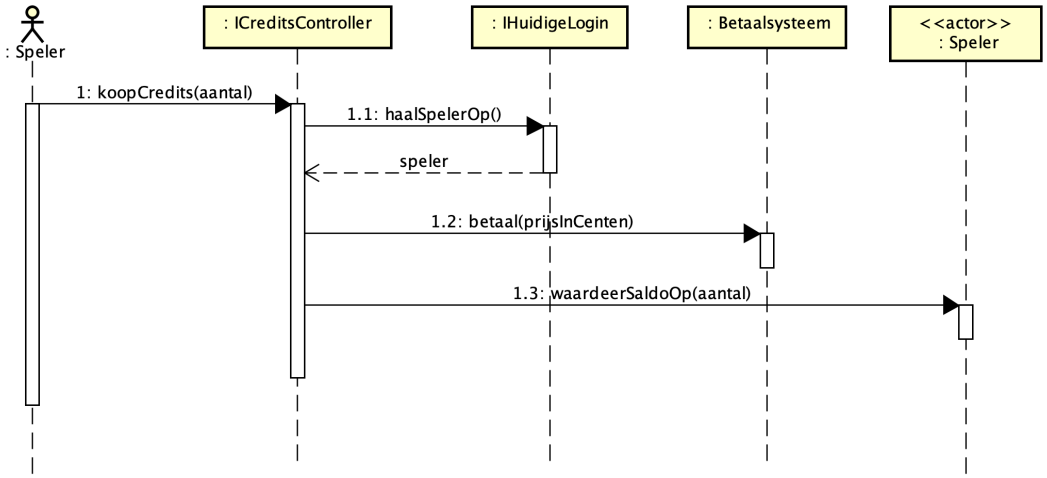
Figuur 6: Sequencediagram voor systeemoperatie *haalScoreOp*



Figuur 7: Sequencediagram voor systeemoperatie *registreer*



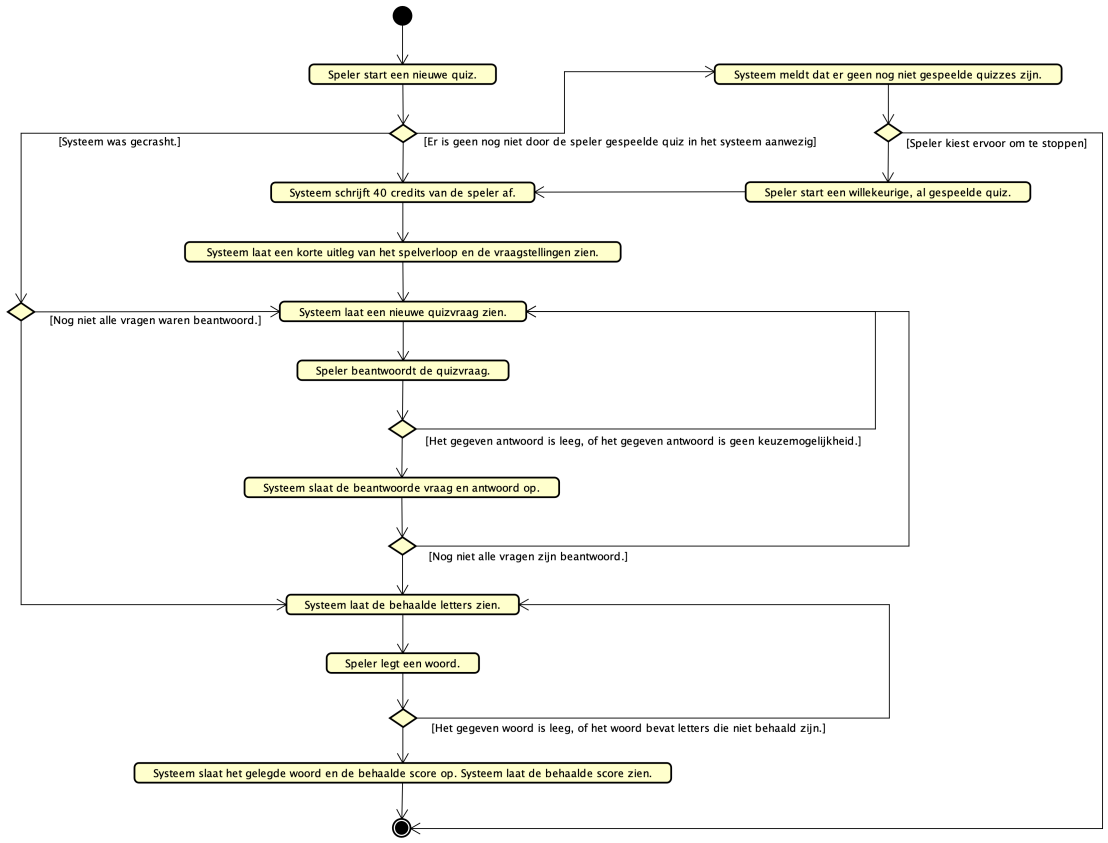
Figuur 8: Sequencediagram voor systeemoperatie *haalAantallenEnPrijzenInCentenOp*



Figuur 9: Sequencediagram voor systeemoperatie *koopCredits*

2.1.3 Activity- en statediagrammen

Deze sectie bevat een activitydiagram voor de usecase *Quiz spelen*.



Figuur 10: Activity diagram van *Quiz spelen*

2.1.4 Gemaakte ontwerpbeslissingen voor subsysteem

Het klassendiagram in figuur 1 bevat een aantal beslissingen gemaakt aan de hand van een aantal design principes en patterns:

- De SOLID principes in tabel 1 (Martin, 2000).
- De Gang of Four design patterns in tabel 2 (Gamma e.a., 1994, p. 97).
- De GRASP principes in tabel 3 (Larman, 2002).

Tabel 1: Gebruik van de SOLID principes in het klassendiagram

Principe	**Plek**	**Beredenatie**
Single Responsibility	Quebble	Deze klasse moet alleen aan te passen zijn wanneer er usecases bijkomen.
Single Responsibility	Vraag	Deze klasse moet alleen aan te passen zijn wanneer er vraagtypes bijkomen.

Wordt vervolgd op de volgende pagina...

Principe	**Plek**	**Beredenatie**
Single Responsibility	ScoreBerekeningStrategy	Deze klasse moet alleen aan te passen zijn wanneer er strategies bijkomen.
Open-closed	Quebble	Quebble geeft slechts een aantal usecase controllers terug. Als er meer usecases bijkomen, hoeft er alleen een nieuwe controller te komen, en moet Quebble een nieuwe methode krijgen. De huidige methodes van Quebble blijven dan intact.
Open-closed	Vraag, ScoreBerekeningStrategy	Door een abstracte klasse te gebruiken is het mogelijk om nieuwe soorten ervan toe te voegen met dezelfde interface. De basisklasse heeft dan een nieuwe factory method nodig, maar de bestaande factory methods kunnen intact blijven.
Liskov Substitution	Vraag	Vraag heeft als subklasses OpenVraag en MeerkeuzeVraag. De interface functioneert exact hetzelfde. Het verschil is dat de Vraag de antwoorden op een andere manier behandelt, en dat een OpenVraag altijd een lege lijst aan mogelijke antwoorden teruggeeft. Dit heeft geen invloed op een gebruiker van de API.
Interface Segregation	I*Controller	Deze scheiden de publieke interface van de concrete klasse. De initialisatie van deze klasse is niet belangrijk voor de publieke API.
Interface Segregation	IQuiz, ISpeler, IVraag	Deze scheiden de publieke interface van de concrete klasse. De concrete klassen bevatten instelmogelijkheden die alleen belangrijk zijn voor het aanmaken ervan. De repositories maken hier alleen gebruik van, de publieke API niet.
Interface Segregation	I*Login*	De AuthenticatieController hoeft de Login alleen maar te beheren. De andere controllers hoeven alleen maar een huidige ISpeler op te kunnen halen.
Dependency Inversion	Quiz	Quiz gebruikt de abstracte klasse ScoreBerekeningStrategy, en maakt daarbij gebruik van diens factory method. Quiz is daardoor niet afhankelijk van de concrete implementatie van de strategy.

Wordt vervolgd op de volgende pagina...

Principe	**Plek**	**Beredenatie**
Dependency Inversion	Quiz	Quiz gebruikt de abstracte klasse Vraag. Een API neemt een lijst met abstracte Vragen aan om in te stellen als attribuut. Quiz is daardoor niet afhankelijk van de concrete implementatie ervan.
Dependency Inversion	Speler	Speler gebruikt de interface IQuiz. Zijn API's nemen IQuizzes aan om in te stellen als attribuut. Speler is daardoor niet afhankelijk van de concrete implementatie van een IQuiz.
Dependency Inversion	QuizBuilder	QuizBuilder gebruikt de abstracte klasse Vraag en diens factory methods. Quiz is daardoor niet afhankelijk van de concrete implementatie van de Vraag.
Dependency Inversion	Quebble	Quebble maakt de Controllers aan, en retourneert de Controllers als I*Controller. De UI klasse is daardoor niet afhankelijk van de implementatie van een I*Controller.

Tabel 2: Gebruik van de Gang of Four patterns in het klassendiagram

Pattern	**Plek**	**Beredenatie**
Builder	QuizBuilder	Ondersteunt het bouwen van quizzes. Allereerst is de API van de Builder ontkoppeld door alleen maar standaard Java types te gebruiken. Quizzes zijn complexe objecten waarin ook Vragen aanwezig moeten zijn, en het moet mogelijk zijn om SpelStatistieken van tevoren in te stellen. Deze builder maakt nog een abstractielaag voor de gebruiker door instanties van IQuiz terug te geven. Dit ontkoppelt de publieke interface IQuiz van de concrete klasse (en de daarbij behorende instelmogelijkheden) Quiz.
Factory method	ScoreBerekeningStrategy	Versimpelt het gebruik van een Strategy. Wanneer de huidige strategy aan vervanging toe is, hoeft een ontwikkelaar alleen maar deze methode aan te passen. De domeinobjecten die afhankelijk zijn van de interface van een strategy, kunnen dan onveranderd blijven.

Wordt vervolgd op de volgende pagina...

Pattern	**Plek**	**Beredenatie**
Factory method	Vraag	Ontkoppelt de interface van een Vraag van de implementaties MeerkeuzeVraag en OpenVraag.
Façade	*Controller	Ontkoppelt de voorkant van de applicatie van de domeinlaag.
Singleton	*Repository	Garandeert dat er maar één instantie is van een Repository. Dit zorgt ervoor dat er altijd één duidelijke bron is van de domeinobjecten, en versimpelt de toegang er naartoe.
Strategy	ScoreBerekeningStrategy	Maakt het makkelijk om het algoritme voor het berekenen van scores later te veranderen.

Tabel 3: Gebruik van de GRASP principes in het klassendiagram

Principe	**Plek**	**Beredenatie**
Creator	ConsoleUserInterface	ConsoleUserInterface maakt direct gebruik van Quebble.
Creator	Quebble	Quebble beschikt over de initialisatie-informatie van de Controllers, namelijk het door de Controllers gedeelde Login object.
Creator	CreditsController	CreditsController is de enige die het Betaalsysteem gebruikt.
Creator	*Repository	De repositories houden de gegevens bij van de corresponderende klassen.
Creator	Spelresultaat	Maakt direct gebruik van ScoreBerekeningStrategy.
Creator	Quiz	Quiz beschikt over de initialisatie-informatie om SpelStatistieken te maken. Quiz gebruikt SpelStatistieken direct.
Creator	Vraag	Vragen beschikken over de intialisatie-informatie om Antwoorden te maken. Vragen gebruiken Antwoorden direct.
Controller	*Controller	De controllers zijn opgesteld als usecase controllers. De API ervan komt zoveel mogelijk overeen met de stappen uit de bijbehorende usecases (zie Software Requirements Specification).
Expert	Quiz	Quiz kan de Spelstatistieken bijhouden omdat hij weet welke Vragen er zijn, kan tellen hoeveel Vragen er goed zijn, en de uitslag van de goedkeuring van het gelegde Woord kan opvragen.

Wordt vervolgd op de volgende pagina...

Principe	**Plek**	**Beredenatie**
Expert	*	Op alle onderstaande plekken waar indirection is toegepast, is die klasse ook de information expert voor alle API's waar hij bemiddelt tussen onderliggende domeinobjecten.
High Cohesion	Vraag	Alle Vraag- en Antwoord klassen zijn heel samenhangend, en het enige toegangspunt als API is hiervoor IVraag.
Indirection	QuizBuilder	Gebruikt standaard Java types als API, en handelt het maken van quizzes met bijbehorende vragen af.
Indirection	*Controller	De controllers zijn façade klassen, en gebruiken enkel standaard Java types als API om de koppeling tussen de domeinlaag en de servicelaag zo los mogelijk te houden. De controllers regelen dan ook de interactie tussen domeinobjecten, zodat deze los gekoppeld blijven.
Indirection	IQuiz	De interface IQuiz gebruikt alleen maar standaard Java types als API, en handelt de interactie tussen zichzelf, Vragen, Woord, SpelStatistieken en Spelresultaat af.
Indirection	Woord	De API gebruikt alleen maar standaard Java types, en Woord handelt de interactie tussen zichzelf en Letters af.
Indirection	Vraag	De API gebruikt alleen maar standaard Java types, en Vraag handelt de interactie tussen zichzelf en bijbehorende Antwoorden af.
Low Coupling	*	Alle klassen die Indirection toepassen dragen bij aan een lage koppeling tussen objecten.
Polymorphism	Vraag	MeerkeuzeVragen en OpenVragen gaan anders om met het goedkeuren van antwoorden, en met het laten zien van mogelijke antwoorden.
Polymorphism	ScoreBerekeningStrategy	Is uitbreidbaar met meerdere factory methods om verschillende varianten aan te bieden.
Pure fabrication	*Repository	Zijn geen onderdeel van het domein, dienen alleen als persistentie hulpmiddel.
Pure fabrication	*Controller, Quebble	Zijn geen onderdeel van het domein, dienen als API voor de voorkant van de applicatie.

Wordt vervolgd op de volgende pagina...

Principe	**Plek**	**Beredenatie**
Pure fabrication	QuizBuilder	Is geen onderdeel van het domein, dient als hulpmiddel om het bouwen van Quiz-objecten te versimpelen.
Pure fabrication	*Login*	Is geen onderdeel van het domein, dient als hulpmiddel om de huidig ingelogde speler bij te houden of aan te passen.
Pure fabrication	ConsoleUser-Interface	Is geen onderdeel van het domein, dient als startpunt en UI voor de applicatie.
Pure fabrication	Stopwatch	Is geen onderdeel van het domein, dient als hulpmiddel om de gespeelde tijd bij te houden.

Referenties

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994, november 10). *Design Patterns: Elements of Reusable Object-Oriented Software* (1ste ed.).

Larman, C. (2002). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (2de ed.). Prentice Hall.

Martin, R. C. (2000). *Design Principles and Design Patterns*. Verkregen 6 september 2015, van https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

Refactoring.Guru. (g.d.). *Inline Class*. Verkregen 9 juni 2021, van https://refactoring.guru/inline-class