Software Design Document: Parola

Auteur: Bram Xanten (1659066), Benjamin v/d Tillaart

Versie: 2 (29/03/24)

Docent: Marco Barion

Vak: OOAD

Inhoudsopgave

[1 Introductie 1](#_Toc162640445)

[1.1 Algemene beschrijving 1](#_Toc162640446)

[1.2 Doel van dit document 1](#_Toc162640447)

[1.3 Definitions, acronyms, and abbreviations 1](#_Toc162640448)

[2 Class diagram 2](#_Toc162640449)

[2.1 Gemaakte keuzes 2](#_Toc162640450)

[3 Sequence Diagrams 3](#_Toc162640451)

[3.1.1 Registreren account UC1 sequencediagram 3](#_Toc162640452)

[3.2 sequence diagram UC2 spelen quiz 4](#_Toc162640453)

[3.3 Sequence diagram geven antwoord 5](#_Toc162640454)

[3.4 Sequence diagram afronden quiz 5](#_Toc162640455)

[4 OO ontwerpprincipes en design patterns 6](#_Toc162640456)

[4.1.1 Patterns 6](#_Toc162640457)

[4.1.2 Code toelichting 6](#_Toc162640458)

# Introductie

## Algemene beschrijving

Dit document bevat het class diagram en de verschillende sequence diagrammen die wij hebben gemaakt voor het kunnen uitprogrammeren van het quizsysteem.

## Doel van dit document

Het doel van dit document is om inzicht te geven in de code van Parola en over de keuzes die wij hebben gemaakt tijdens het ontwerpen van deze code.

## Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| Term | Description |
|  |  |

# A diagram of a computer Description automatically generated with medium confidenceClass diagram

De meeste getters en setters hebben wij achterwege gelaten om het diagram leesbaar te houden. De belangrijkere getters die niet standaard gegenereerd kunnen worden zijn wel nog in het diagram verwerkt. Een voorbeeld hiervoor is getAnswers omdat dit een lijst aan answers meegeeft en niet maar een answer.

## Gemaakte keuzes

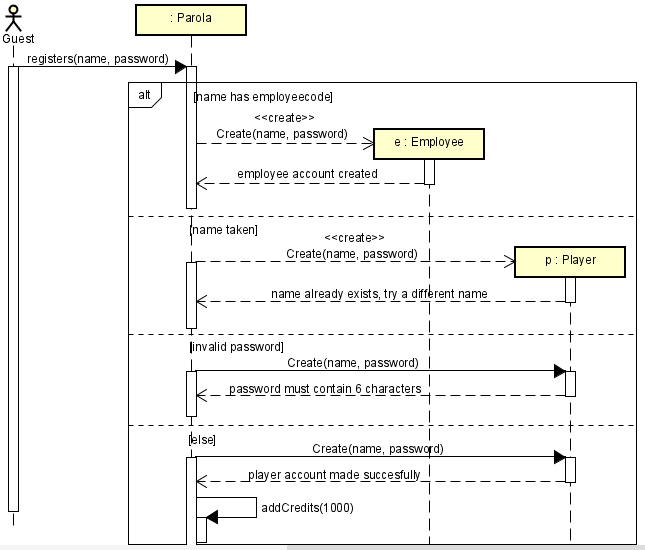
Wij maken gebruik van het adapter pattern voor beide het betalen van credits in de winkel en voor het controleren van het woord. Voor beide deze onderdelen moeten wij gebruik maken van externe onderdelen en hierom hebben wij er ook voor gekozen om het adapter pattern te kiezen. Dit zorgt er ook voor dat als er een wijziging is in de externe code dat wij niet velen wijzigen hoeven te maken in een van onze belangrijkere/grotere classes. Zo hoeven wij alleen de adapter te wijzigen of te koppelen aan een andere externe class om de koop snel weer functioneel te maken. Dit is natuurlijk hetzelfde bij het extern laten controleren of een woord bestaat.

Een andere keuze was om alle class variabelen private te maken. Dit zorgt voor meer information hiding. Waardes die niet aangepast mogen worden vanuit andere classes hebben ook geen setters.

In hoofdstuk 2.2 zijn nog een aantal keuzes beschreven in combinatie met de GRASP en SOLID principes.

# Sequence Diagrams

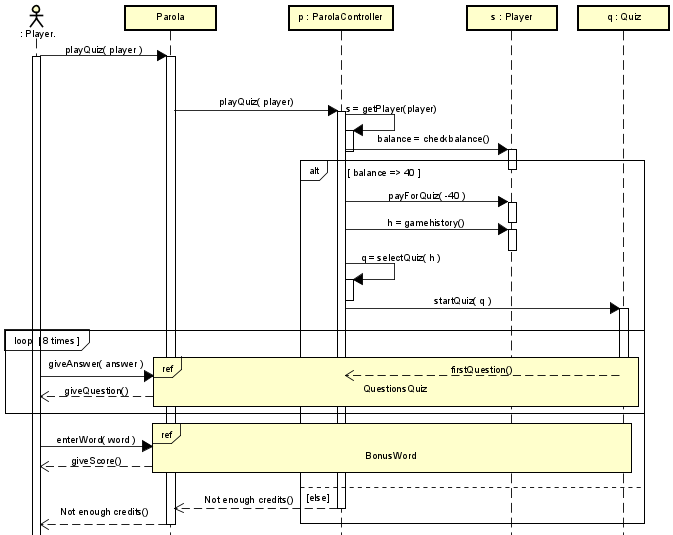
## Registreren account UC1 sequencediagram



Bij het sequence diagram hierboven is te zien hoe het registreren van een gast gebruiker plaats vind. Zoals in UC1 of de fully dressed use case te zien is in het SRS of het system sequence diagram ervan. Kan een gast een aanvraag doen om te registreren. Hij of zij vult dan zijn naam en paswoord in. Als de naam een medewerkercode heeft wordt een medewerker account aangemaakt. De code is als beveiliging dat niet een random speler een medewerker kan worden.

Om een speler te worden moet het wachtwoord 6 characters lang zijn en de naam een unieke naam zijn die niet eerder voorkomt. Als het account succesvol is aangemaakt krijgt de speler een startsaldo van 1000 credits om quizen mee te spelen.

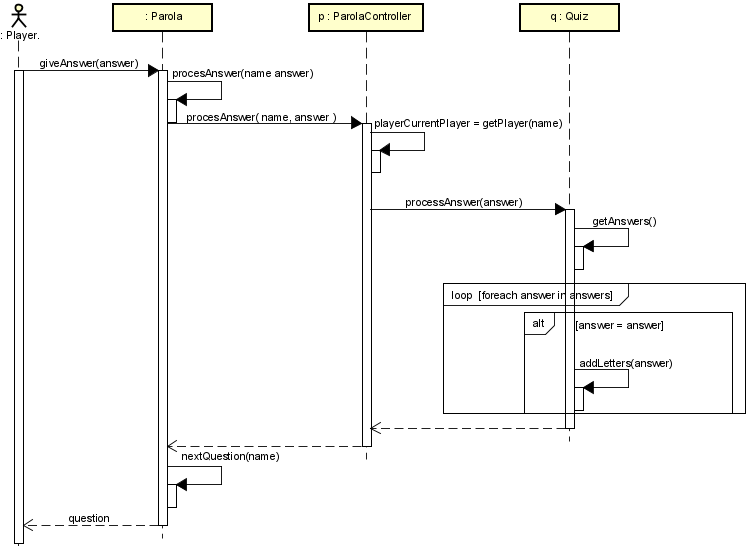
## sequence diagram UC2 spelen quiz

Bij spelen quiz is een speler al van te voren ingelogd zoals beschreven in de fully dressed use case. Zodra een speler ervoor kiest om een quiz te spelen wordt de functie eerst doorgegeven aan de parolaController zodat information expert wordt toegepast. Hier wordt de speler opgehaald omdat de controller een lijst heeft van alle spelers.   
Omdat elke quiz die je speelt 40 credits kost moet worden gekeken wat het huidige balans is van deze speler. Als deze hoger is of gelijk aan 40 halen we er 40 vanaf. Ook willen we zorgen dat de speler zo min mogelijk dezelfde quiz speelt. Daarom halen we ook de geschiedenis op voordat we een quiz selecteren.

Omdat we de controller alles laten delegeren en niet de speler een quiz laten starten passen we het single responsibility toe. Zodra de quiz is gestart zal het systeem de eerste vraag weergeven aan de speler waarna deze een antwoord erop geeft.

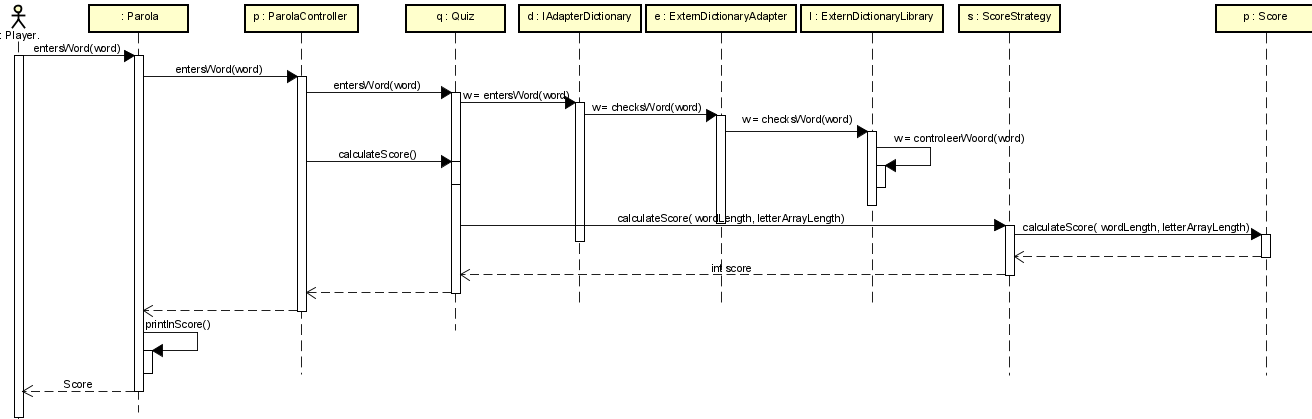
Er mag maar 1 input per sequence diagram worden weergeven daarom gaat deze return naar een ref wat een andere sequence diagram is die de vragen afhandeld. Daarna een derde die het bonuswoord afhandeld. Pas daarna kan de lifeline van spelen quiz eindigen en hebben wij gekozen om dit op deze manier weer te geven in een sequence diagram, zodat een makkelijke representatie van het hele proces zichtbaar is.

## Sequence diagram geven antwoord



Dit is het vervolg van sequence diagram van 2.3. waar de speler de eerste vraag krijgt van de quiz. De speler geeft dan antwoord op de vraag die wordt doorgegeven naar de quiz. Hier wordt het vergeleken met de lijst van antwoorden of het een goed antwoord is. Als het antwoord juist is wordt er een letter bijgehouden voor de speler die deze allemaal krijgt nadat de vragen beantwoord zijn. Omdat deze hele sequence diagram in een loop staat wordt dit 8 keer herhaald totdat alle vragen van de quiz zijn beantwoord.

## Sequence diagram afronden quiz

Nadat alle vragen zijn beantwoord kijgt de speler de letters van alle juiste antwoorden om een woord te vormen. Dit woord wordt dan doorgegeven naar de externe library om single responsibility te waarborgen. Nadat het woord gecontroleerd is wordt de score berekent op basis van de lengte van het woord en de aantal juiste antwoorden. Deze worden doorgegeven aan bereken score die het dan via het juiste strategy pattern uitrekent.

Deze score wordt dan als int terug gegeven aan de main die het uitprint voor de speler om te zien. Daarna is het spelen van de quiz afgelopen, sequence diagram 2.3.

# OO ontwerpprincipes en design patterns

**Single responsibility** hebben wij zoveel mogelijk proberen toe te passen in de applicatie. Bij de parolaController was dit niet altijd mogelijk omdat we vanuit een voorgedefineerde main moesten werken en de controller veel connecties heeft met de andere klassen. Zo houden wij het aantal spelers en de taal bij wat nog opgesplitst kan worden.

**Creator pattern** wordt bij Parola alleen gebruikt als een medewerker een nieuwe quiz of vragen aanmaakt.   
Information expert hebben wij bijna overal toegepast. Zo worden verantwoordelijkheden doorgegeven tot deze bij de juiste klasse aanwezig zijn die hier iets mee moet doen.

### Patterns

**Adapter pattern:** Voor de externe library en voor het betaalsysteem hebben wij gekozen om een adapter pattern toe te passen. De reden hiervoor is omdat wij niet weten hoe de banken hun code hebben geschreven door een adapter klasse te maken kunnen wij alsnog onze informatie naar hen toe sturen en informatie terugontvangen.

Dezelfde reden voor het externe library systeem waar we het bonus woord naar toe sturen om te kijken of het een woord is dat voorkomt in het woordenboek. Door het steeds' door te sturen maken wij ook gebruik van het single responsibility principe. Zoals te zien is in sequence diagram 2.5

We zaten eraan te denken om een **factory pattern** ervoor te zetten, maar we weten niet hoeveel verschillende betaalservices Parola gaat krijgen dus we vonden de uitbreiding het niet waard op dit moment om het overzicht meer chaotisch te maken.   
  
**Strategy pattern** hebben wij voor gekozen zodat wij makkelijk kunnen overswitchen naar een andere manier van score berekenen zonder de code aan te passen. Dit zorgt ervoor dat wij het open closed principe aanhouden. Dan kunnen wij nieuwe methoden voor punten berekenen toevoegen zonder te veel aan te passen.

### Code toelichting

Bij het einde van een quiz zal de score moeten worden berekend mede aan de hand van de lengte van het geraden woord. Hierbij is besloten om het raden van een woord nog onderdeel uit te laten maken van de quiz. Hierom heeft de quiz ook een dependency aan IAdapterDictionary. Nadat het woord is gecontroleerd en hij blijkt te bestaan zal de lengte van het woord worden doorgestuurd voor de scoreberekening. Indien het woord niet bestaat wordt de waarde 0 meegegeven.

In de timer class wordt er gebruik gemaakt van “java.util.concurrent.TimeUnit”. Dit wordt gebruikt voor het omzetten van de millisecondes die wij verkrijgen door het gebruik van getCurrentMillis naar secondes.