

**Object Georiënteerd Ontwerp**

**Bestel App Broodjeszaak**

Coddens Jasper

Schaaf Louis

Schreppers Jarne

Thomas Slegers

2021, Toegepaste Informatica

26 december 2021

University Colleges Leuven-Limburg  
Campus Proximus  
Geldenaaksebaan 335  
3001 Heverlee  
Tel. +32 (0) 16 375 700

# Algemene opmerking

Dit verslag gaat over de opdracht ‘Bestel app broodjeszaak’, dat zal meetellen voor 5 punten van je totaalscore van dit vak. Dit verslag vormt de toelichting bij jouw code.

**Je wordt geacht om je voor het verslag volledig aan deze template te houden. Buiten deze ‘algemene opmerking’ zijn alle paragrafen verplichte onderdelen van het verslag.**

**Je maakt een zip file van al je source code (.java bestanden, geen .class bestanden) en van alle bestanden die van belang zijn voor deze OOO opdracht. Je voegt tevens de laatste versie van je verslag (Word document) toe aan deze zip file.**

**Je laadt deze zip file op via Toledo uiterlijk op woensdag 29/12/2021 - 23.59 uur. Naam van de zip file: volgnr\_familienaam1\_familienaam2\_familienaam3\_familienaam4\_Broodjeszaak\_2021\_2022 De familienamen zijn oplopend alfabetisch geordend (vb 17\_Cox\_Dox\_Jansen\_Lenaerts\_ Broodjeszaak\_2021\_2022).**

**Het spreekt voor zich dat de code op Toledo overeenstemt met de code in je repository op GitHub.**

# URL GITHUB repository

**Copy/paste hier de URL van je Github repository bestel app project**

# URL

https://github.com/Louis-schaaf/OOgroep17

# Vereisten

Geef voor het gehele project aan welke vereisten (evt. zelf verder uitgewerkt/opgesplitst aan de hand van de opgave) je succesvol hebben geïmplementeerd, en welke topics niet gelukt zijn. Indien je een deel van de voorziene vereisten niet afgewerkt hebt, geef dan aan waarom niet… De reden kan “tijdsgebrek” zijn, het kan een issue zijn “wist niet hoe, het crashte”, of het kan zijn dat je een zeer goede reden had om het niet te implementeren…

Als er zaken zijn uit de opdracht die je niet hebt kunnen uitwerken of die je beter zou willen uitwerken, dan som je deze hier tevens op (in laatste rij van tabel).

|  |  |  |
| --- | --- | --- |
| User story | OK? | indien niet ok – wat lukt niet en waarom? |
| 1. Overzicht broodjes/beleg tonen | OK |  |
| 1. Overzicht broodjes/beleg tonen met keuze uit tekst of excel | OK |  |
| 1. Een nieuwe bestelling opstarten | OK |  |
| 1. Broodjes/beleg toevoegen | OK |  |
| 1. Identieke bestellijnen toevoegen | OK |  |
| 1. Bestellijn Wissen   Bestelling annuleren | OK |  |
| 1. Bestelling afsluiten | OK |  |
| 1. Bestelling betalen en doorsturen naar keuken | OK |  |
| 1. Bestelling opvragen en afwerken in keuken | OK |  |
| 1. Aanpassen settings | OK |  |

# 

# Design patterns

Geef voor elk gezien patroon aan waar je het gebruikt hebt (mogelijkerwijs meer dan eens). Genereer een klassendiagram voor elke situatie waarin je het patroon gebruikt hebt. Geef extra informatie (voordelen / waarom / …). Als je een patroon niet toegepast hebt, leg uit waarom niet.

|  |  |  |
| --- | --- | --- |
|  | Toegepast (ja/nee)  In welke stories(nr)  Waarom toegepast(voordeel) | Bijhorend klassendiagram (gegenereerd vanuit je java code) |
| Observer | Ja, story 3, story 4, story 5, story 6  De BestelFacade dat in ons geval de observable, subject is van de BestelViewController, KeukenViewController, InstellingenController, OverzichtController en de StatistiekenController. Het voordeel is dat alle controllers worden geupdate vanaf er iets verandert aan de bestelling en op de hoogte blijven van elkaar. | Kijk naar bijlage 1 |
| Strategy | Ja, Story 1, 2, 7  In story 2 hebben we een load/save methodes gemaakt voor een excel bestand te kunnen bewerken. De reden waarom is omdat we in story 1 tekst load/save methodes gemaakt hebben. Het was dus belangrijk dat de applicatie flexibel bleef omdat bij het starten van de applicatie de juiste strategy kiest, het zorgt ook dat we makkelijk nieuwe strategy’s kunnen toevoegen in de toekomst. Belegdatabase en Broodjesdatabase gebruiken beide de load/save strategy’s. We gebruiken ook een kortingstrategy, waar al verschillende strategy’s van mogelijk zijn maar door strategy mogelijk zijn meer van toe te voegen op een later moment | Kijk naar bijlage 2.A en 2.B |
| Simple Factory | Ja story 1, 2, 7  De strategy’s maken gebruik van factory’s. Dat is dus de KortingStrategyFactory en de LoadSaveStrategyFactory. We willen niet dat de context klasses weten welke strategy wordt geinstancieerd. De factory zal dus de juiste strategy bepalen en instantieren. Hierdoor moeten de context klasses ook maar 1 methode gebruiken voor de juiste strategy te implementeren. | Kijk bijlage 2.A en 2.B |
| Façade | Ja, story 4  Onze BestelFacade klasse is de façade klasse van Bestelling. De BestelFacade weet alles over het subsysteem Bestelling. De controllers spreken dus BestelFacade aan voor alles dat ze willen weten over bestelling en weten niet wat in de achtergrond wordt gedaan zo hebben de controllers geen besef van alle andere klasses in het model. | Kijk bijlage 1 |
| Singleton | Ja, story 1, 2 en 7  Bij onze factory’s gebruiken we het singleton pattern. Dat betekent dat een factory maar 1 keer wordt aangemaakt en deze zal blijven gebruiken door het statisch te maken.  Het voordeel hiervan is bij een factory dat die maar 1 keer gebruikt moet worden voor de strategy te selecteren. | Kijk bijlage 2.A en 2.B |
| State | Ja, story 3  We gebruiken de BestellingState Interface voor Bestelling. Dit toont al de verschillende states dat bestelling kan nemen. Het gedrag van Bestelling verandert met de verschillende states. Zo is er volledige controle over de bestelling en over het gedrag van de bestelling in al de verschillende states. De code wordt ook zo veel beter te beheren. We minimaliseren de if statements met states te implementeren wat veel slordiger is en onoverzichtelijker. | Kijk bijlage 3 |
| MVC | Ja, het volledige project. Elke story is geimplementeert met het MVC patroon. We hebben dus 3 packages: de model, de view en de controller. Het zorgt voor een overzichtelijker project. Je kan alles sneller terug vinden en het is veel leesbaarder. | Kijk bijlage 4 |
| Template method | Story 1  We gebruiken voor onze load/save functionaliteiten van tekst bestanden en excel bestanden een template klasse die we uitbreiden in een aparte klasse. Het voordeel hiervan is dat we sommige methodes niet kunnen aanpassen die ook niet aangepast moeten worden en je hebt minimale dubbele code door de super klasse. | Kijk bijlage 5 |

# Speciale topics

Geef voor elke “speciale topic” aan of je het gebruikt hebt of niet, en zo ja, waar. Toon eventueel aan met een klassendiagram.

|  |  |  |
| --- | --- | --- |
|  | Toegepast (ja/nee)  In welke stories(nr)  Waarom toegepast(voordeel) | Bijhorend klassendiagram of extra uitleg |
| Enum | Ja Story 1, 2, 3  (excel, korting, bestellingevents)  We gebruiken LoadSaveStrategyEnum, KortingStrategyEnum en BestellingEvents. Bij nieuwe strategy’s toe te voegen zou alleen de enums moeten verandert worden zonder veel refactoring dit is hetzelfde voor bestellingevents als we een nieuwe event bij de bestelling zouden toevoegen. | Kijk bijlage 2.A en 2.B |
| Properties | Ja het hele project (de juiste strategy)  Als de applicatie opstart zal de propperties worden ingelezen en de juiste configuratie aannemen met de juiste strategy’s. Zodat als je de applicatie afsluit de propperties worden onthouden en bij de volgende keer opstarten terug worden getoont. | Kijk alle klassendiagrammen |
| Reflection | Ja Story 1, 2 en 7  We gebruiken reflection bij beide factory classes, dus bij KortingStrategyFactory en LoadSaveStrategyFactory. Het voordeel is dan dat we geen oneindige if statements moeten gebruiken voor het controleren dat we de juiste klasse gebruiken. De factory moet dan ook niet weten wanneer een nieuwe strategy wordt toegevoegd. | Kijk bijlage 2.A en 2.B |
| Andere… | Nee |  |

# Werkverdeling

Geef aan in percentages hoeveel je bij benadering gespendeerd hebt aan deze opdracht.

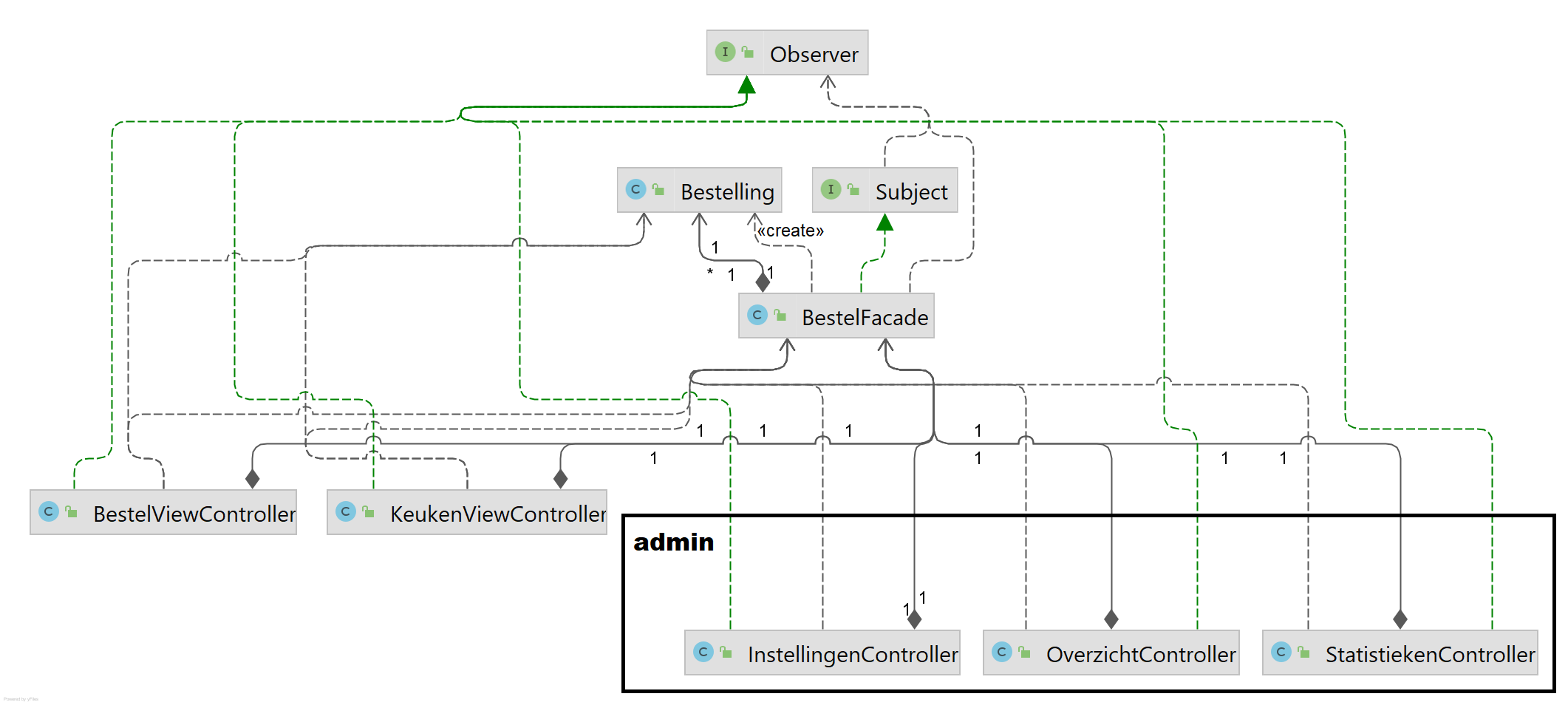
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Coddens | Schaaf | Schreppers | Slegers | Totaal |
| Ontwerp | 20% | 30% | 20% | 30% | 100% |
| Implementatie | 30% | 20% | 30% | 20% | 100% |
| Verslag | 25% | 25% | 25% | 25% | 100% |

# Klassen diagramma’s

Voeg het definitieve gegenereerde klassendiagram van je code toe.

**Dit klassendiagram moet goed leesbaar zijn.** Spreid het over meerdere pagina’s (bv. 1 blad per package (MVC) en 1 overzicht klassendiagram (zonder attributen en methoden). GEEN KLASSENDIAGRAM MET ZWARTE ACHTERGRONKLEUR AFPRINTEN!!!!!!!!!!

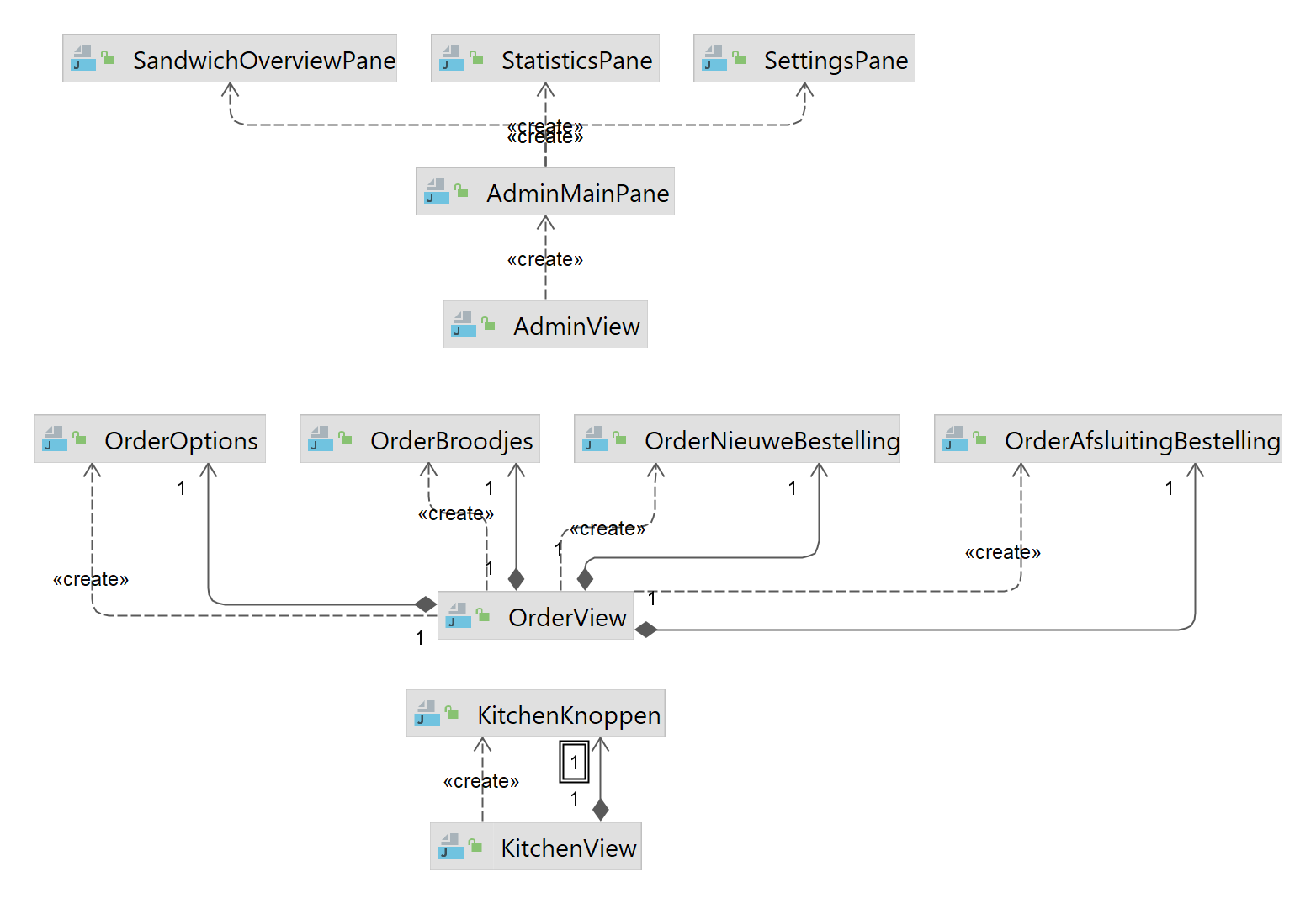
**Package controller:**

****

**Package model:**

Diagram

Description automatically generated

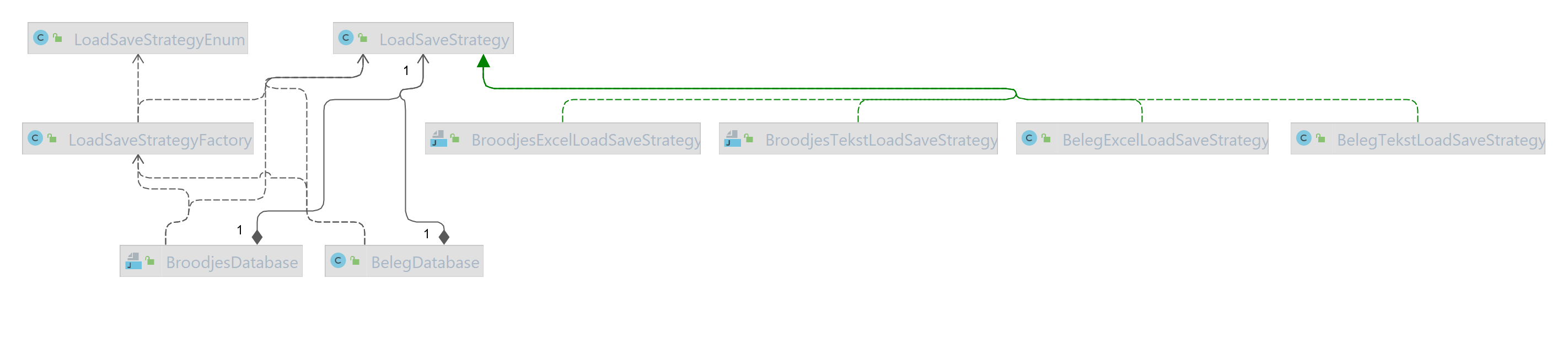
**Package view:**

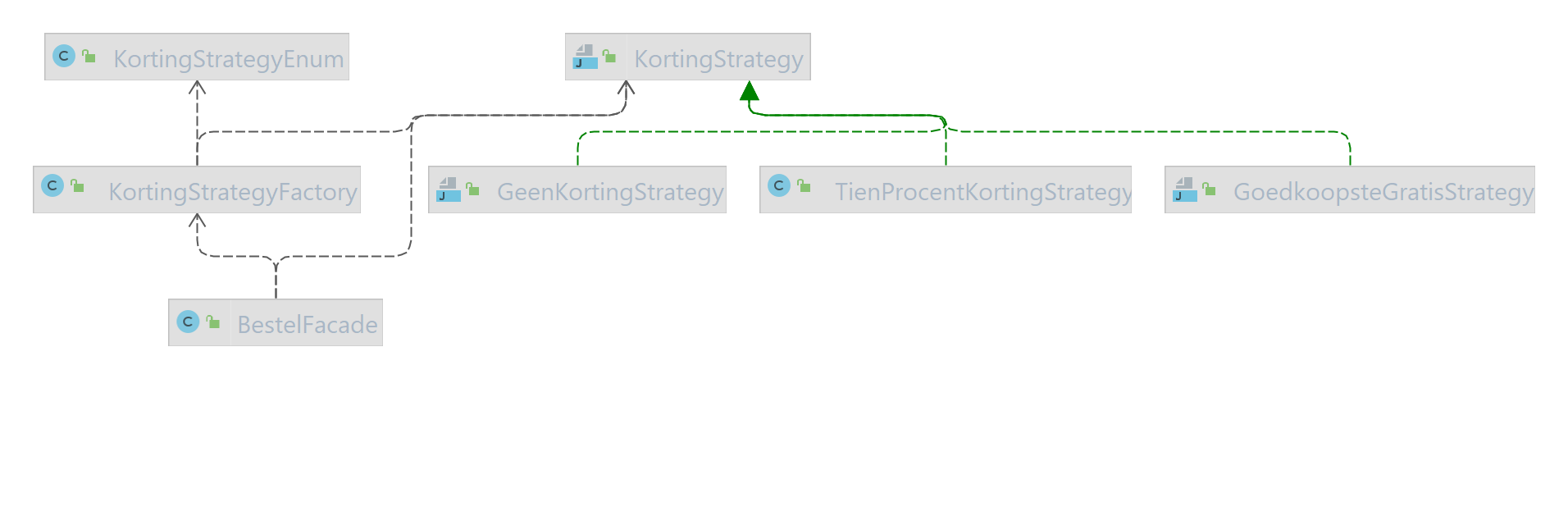
**Bijlage 1:**

Diagram

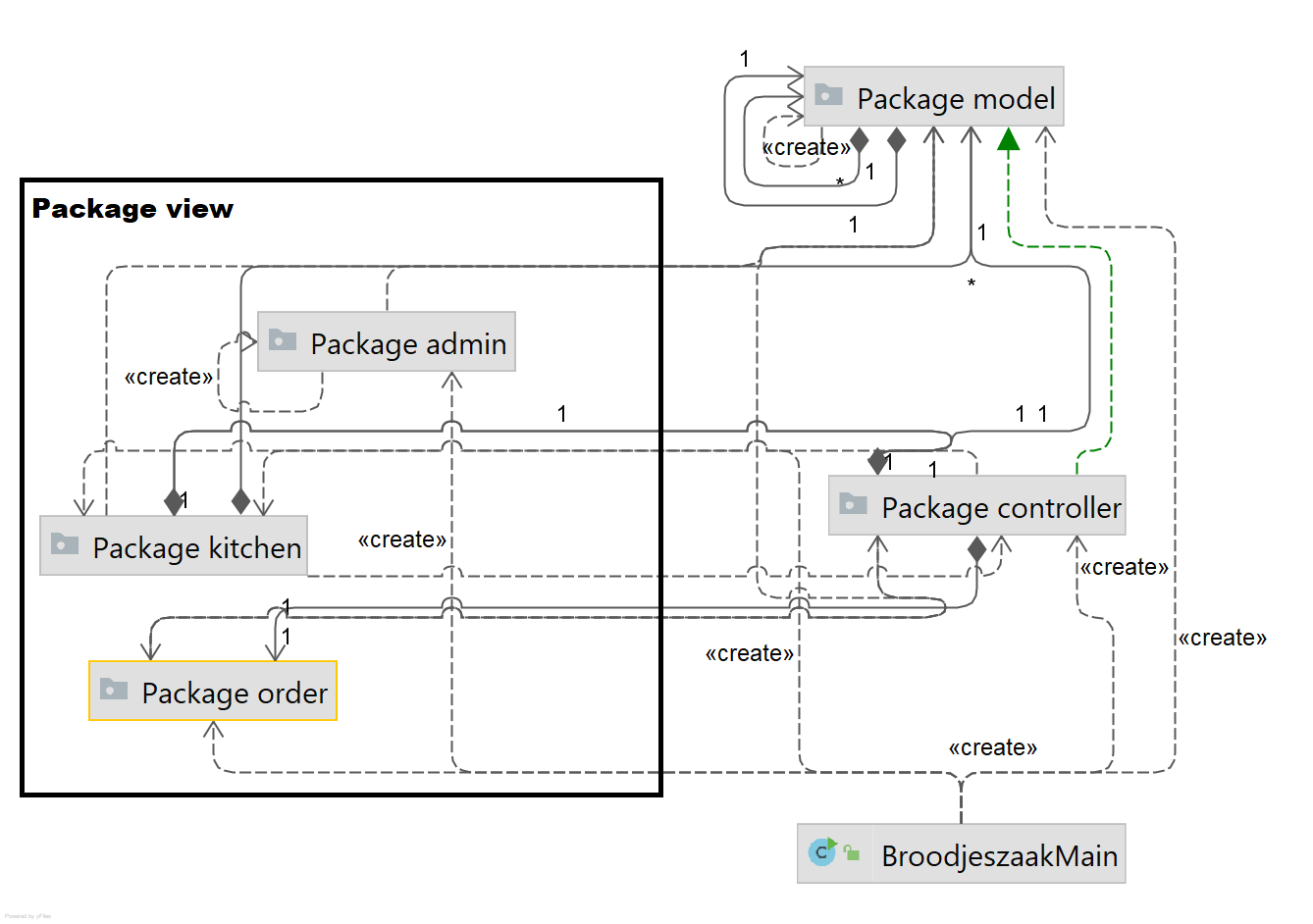
Description automatically generated

**Bijlage 2.A:**

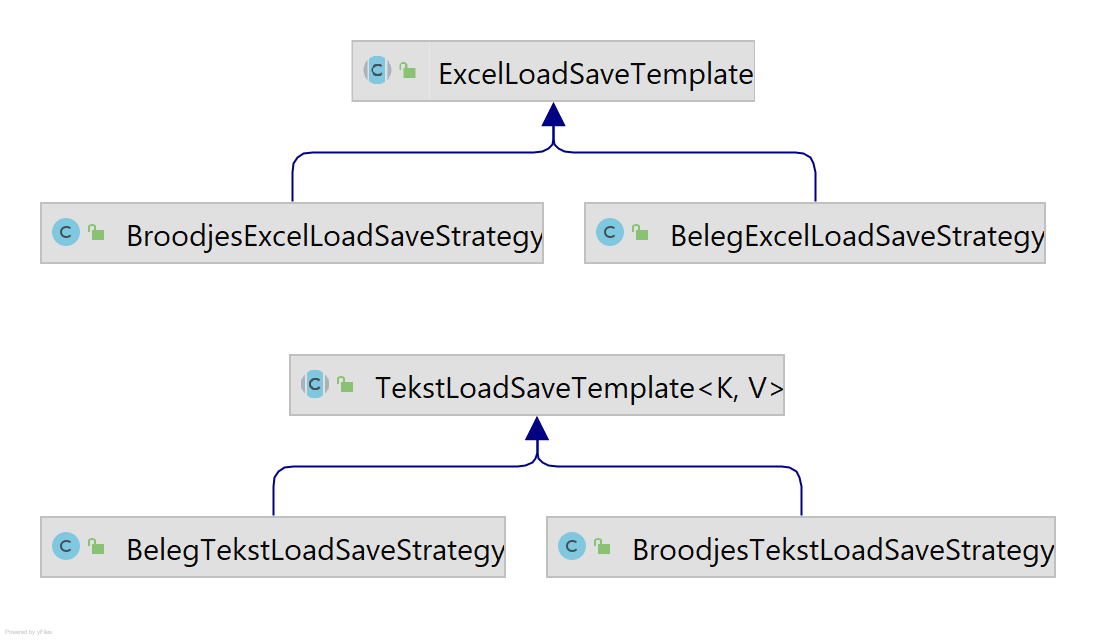


**Bijlage 2.B: Bijlage 3: Diagram

Description automatically generated**

**Bijlage 4:**

**Bijlage 5:**

****