

**Object Georiënteerd Ontwerp**

**Gokspel App**

Gataev Adam

Mustafa Jurgen

Reintjens Frits

2020-2021, Toegepaste Informatica, VT

24/05/2021

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

# Algemene opmerking

Dit verslag gaat over de opdracht ‘Gokspel app’, 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 maandag 31 mei 2021 - 23.59 uur. Naam van de zip file: volgnr\_familienaam1\_familienaam2\_familienaam3\_** **GokSpel\_2020\_2021\_V. Het volgnr krijg je van de lector.**

**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 gokspel app project**

# URL: <https://github.com/FritsReintjens/3_Gataev_Mustafa_Reintjens_GokSpel_2020_2021_V>

# 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 spelers tonen | Ja |  |
| 1. Overzicht spelers tonen met keuze uit tekst of Excel formaat | Ja |  |
| 1. Speler meldt zich aan om spel te spelen | Ja |  |
| 1. Een aangemelde speler kiest een gokstrategie | Ja |  |
| 1. Een aangemelde speler werpt de dobbelstenen | Ja |  |
| 1. Een aangemelde speler wijzigt na de tweede worp zijn inzet | Ja |  |
| 1. De spelsessie statistieken worden bijgewerkt | Ja |  |
| 1. De speladmin initieert het spel voor een nieuwe speler/beëindigt de spelsessie | Ja |  |
| 1. De speladmin wijzigt de settings van het gokspel | Ja |  |

# 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, in story 3,5,7,8,9. Hierdoor hebben wij verschillende zaken kunnen updaten (in real time) |  |
| Strategy | Ja, in story 2,4.  Als je bijvoorbeeld de gokstrategies neemt, alle strategies doen hetzelfde: je kan een beschrijving opvragen, winst opvragen, ....  Door al deze strategies te laten implementeren van een GokStrategy interface kan je eenvoudig weg gokStrategy.getWinst() doen ipv if statements te moeten gebruiken om te kijken op welke klasse je .getWinst() moet doen. |  |
| Simple Factory | Ja, in story 2.  Het voordeel van factory is dat je de factory kan aanspreken om een nieuw object aan te maken.  Als we GokStrategies bekijken hadden we eerst 3 strategies. Als we een nieuwe strategie willen toevoegen hoeven we nu maar de juiste info in de GokStrategyEnum toe te voegen en de factory lost het aanmaken van het object op. |  |
| Façade | Ja in story 3 (technisch gezien overal).  De façade is de link naar alle methodes en variabelen in de model klassen die moeten gebruikt of gewijzigd worden. Hierdoor hoeven we niet steeds te gaan graven in de verschillende klassen. |  |
| Singleton | Ja in story 2.  Hierdoor zijn we zeker dat we maar 1 object hebben van de klasse (in dit geval onze database) en dat alle andere klassen interageren met dit ene object. |  |
| State | Ja, in story 6 (begonnen bij story 4 en 5).  Aan de hand van states kunnen we bepaalde methodes aanroepen die van belang zijn in die toestand. Hiermee voorkomen we ook dat we geen methodes oproepen die niet mogen opgeroepen worden. |  |
| MVC | Ja, in story 3 (technisch gezien overal).  Dit zorgt ervoor dat de view is afgescheiden van de model klassen en de controllers de link zijn tussen de twee |  |
| Template method | Ja, in story 1.  Hierdoor moeten wij niet telkens een nieuwe methode maken voor iedere functionaliteit doordat wij stukjes code gaan hergebruiken. |  |

# 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, in story 2.  We hebben enums als een link tussen Strategies en Factory. |  |
| Properties | Ja, in story 9.  Het voordeel van properties is dat je werkt met key value paren. En je kan simpel schrijven en lezen met de properties-library. | property=value  …  Strategies=EvenOgenStrategy\:5,SomOgenStrategy\:10  LoadSaveFormat=TEKST |
| Reflection | Ja, in story 2 en 4.  Wij hebben dat in onze factory klasses gebruikt.  Het programma zal dus in runtime de klassenaam inspecteren zonder deze te kennen wanneer het programma compiled wordt. | Class dbClass = Class.forName(klasseNaam);  loadSaveStrategy = (LoadSaveStrategy) dbClass.newInstance(); |
| Andere… |  |  |

# Werkverdeling

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

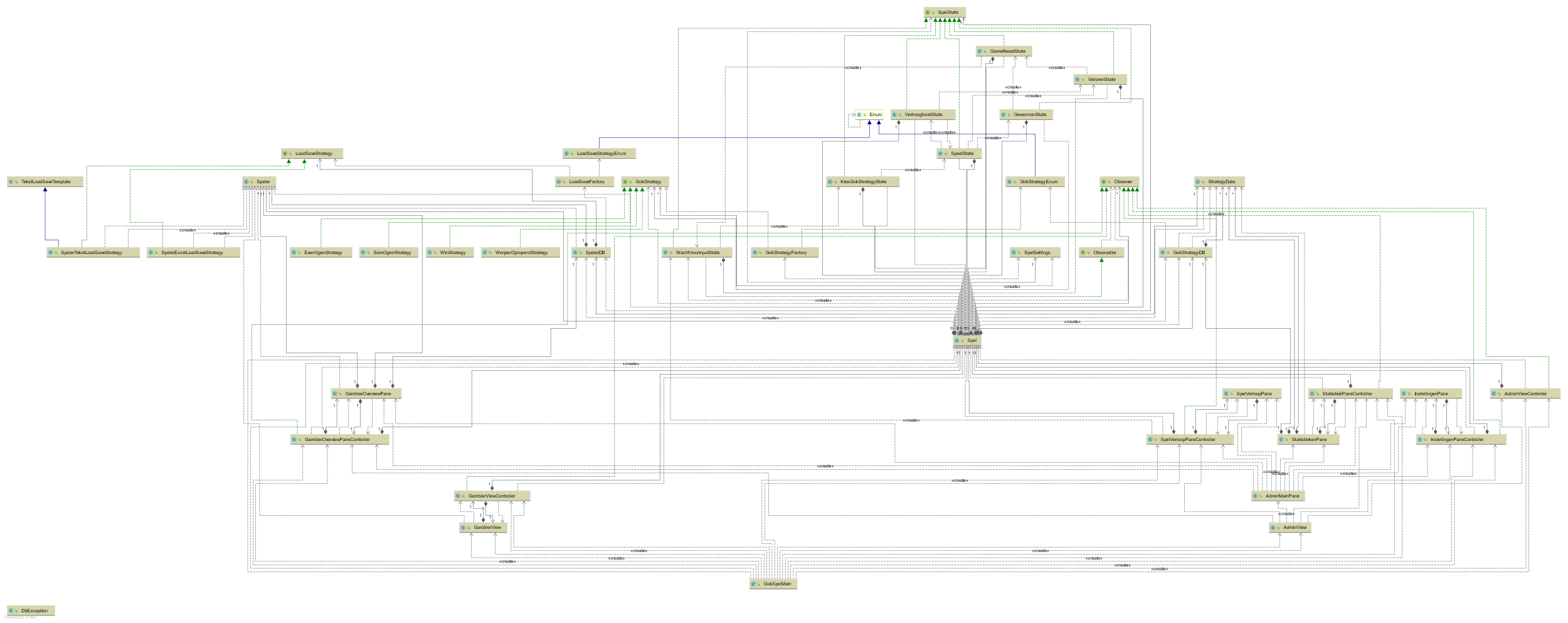
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Gataev | Mustafa | Reintjens | Totaal |
| Ontwerp | 35 | 30 | 35 | 100% |
| Implementatie | 35 | 30 | 35 | 100% |
| Verslag | 33 | 33 | 33 | 100% |

# Klassen diagramma’s

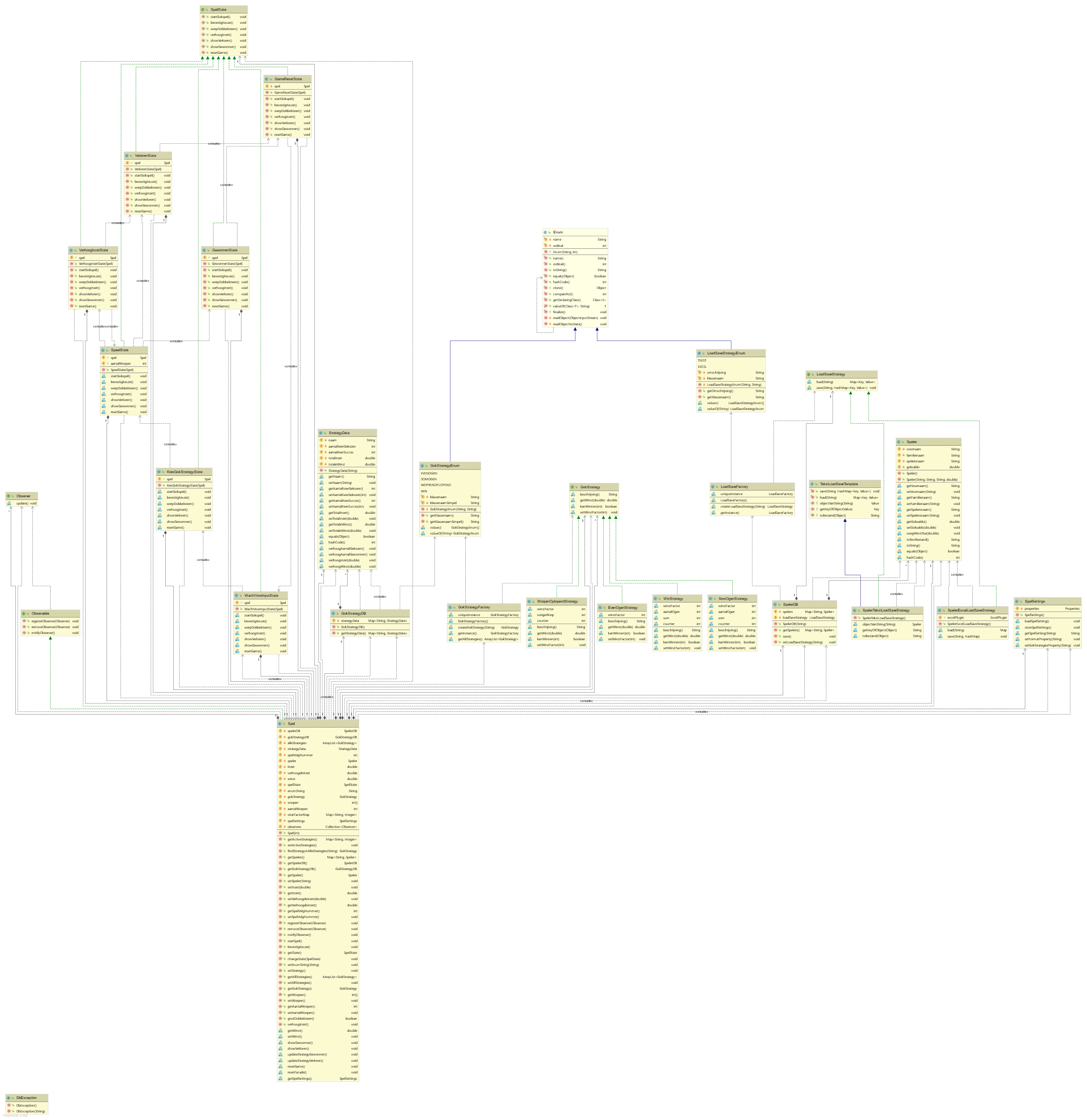
Voeg het definitieve gegenereerde klassendiagram van je code toe.

**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!!!!!!!!!!

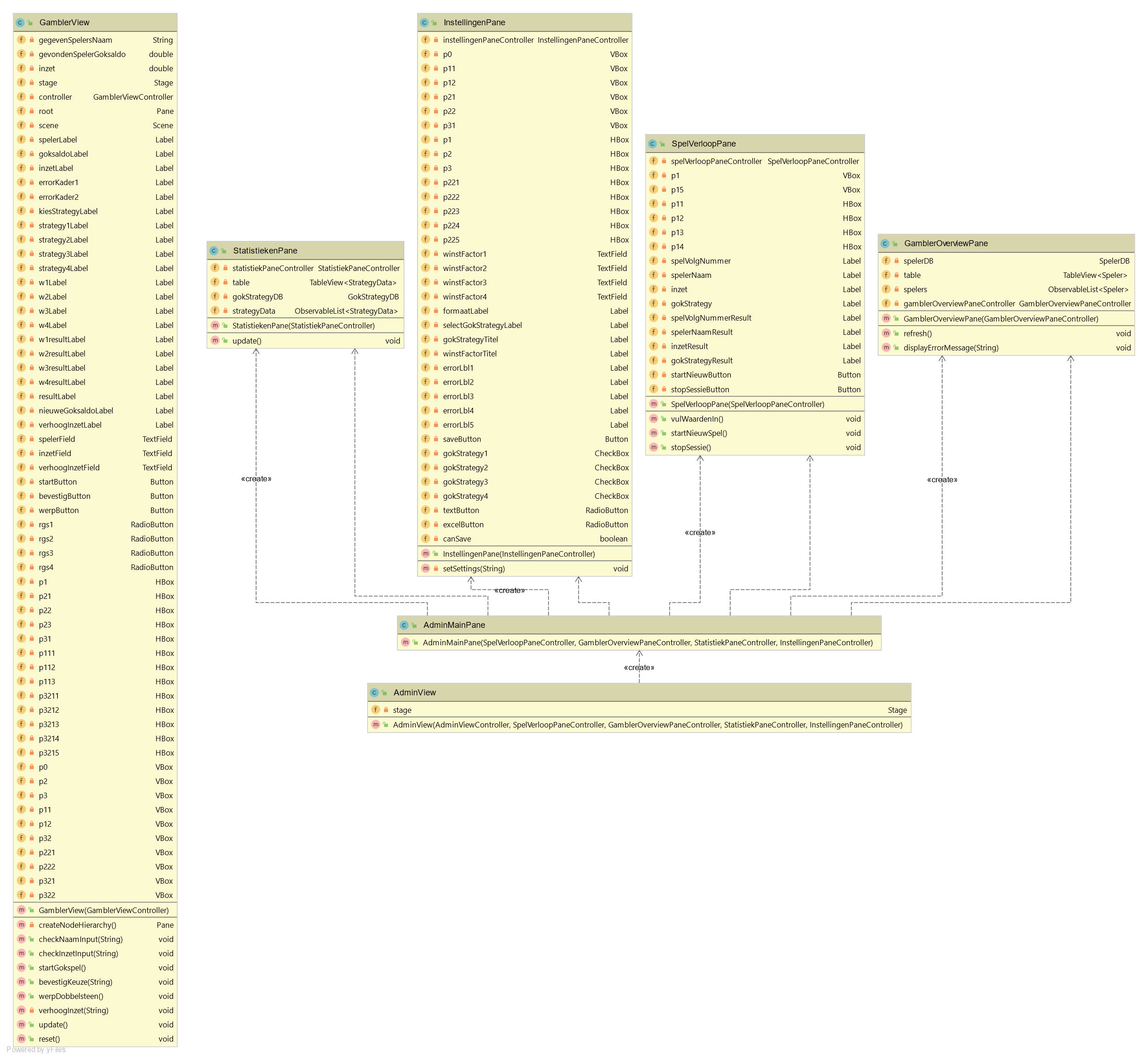
Alles:



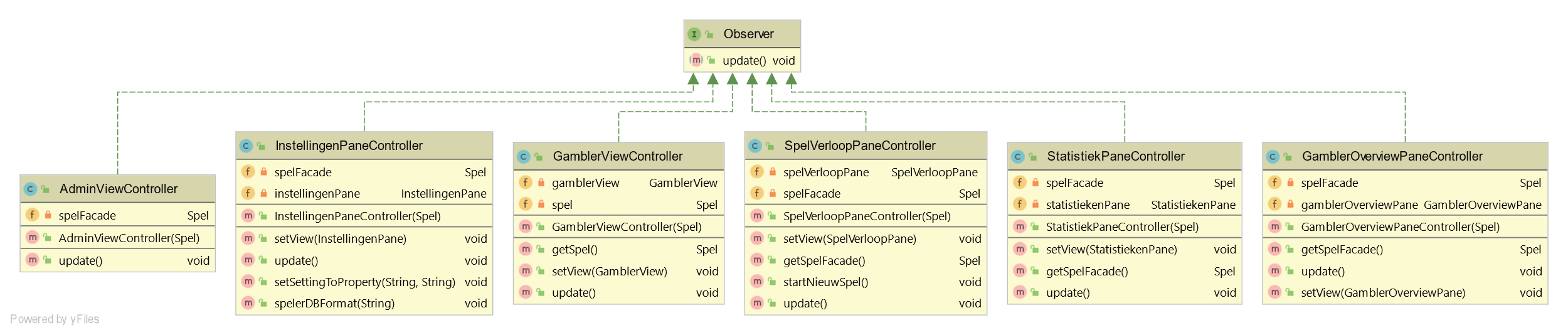
Model:



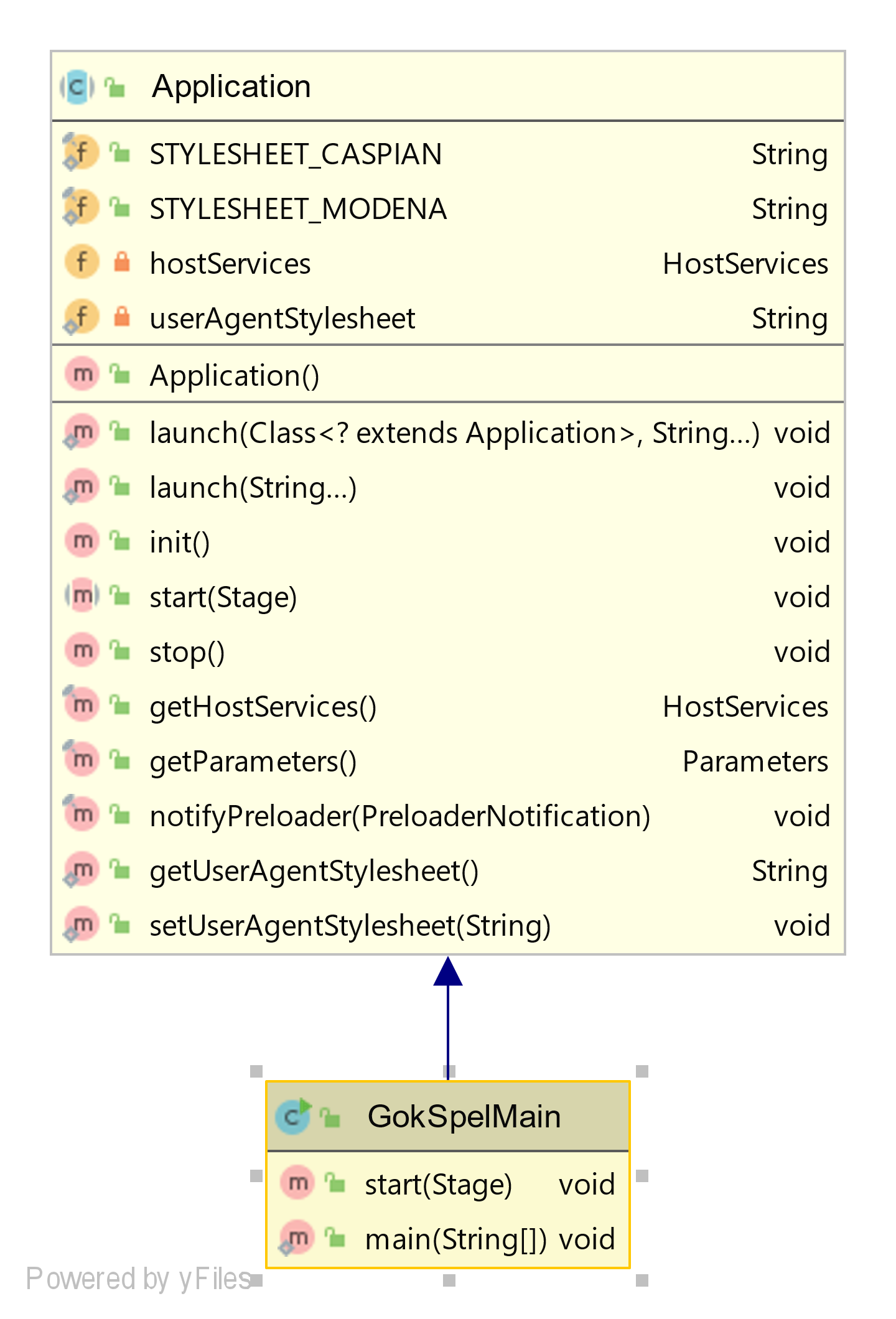
View:



Controller:



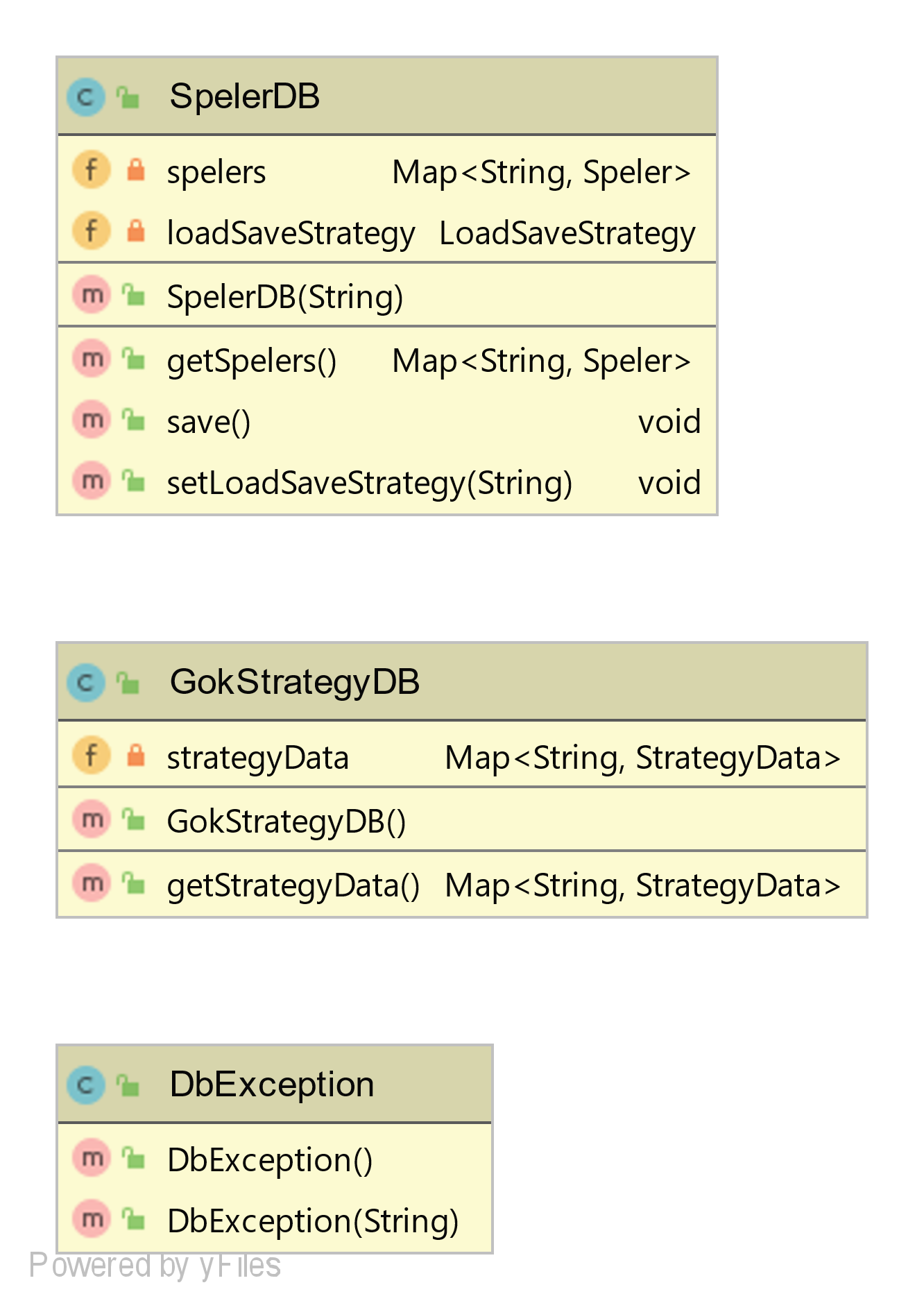
Application:



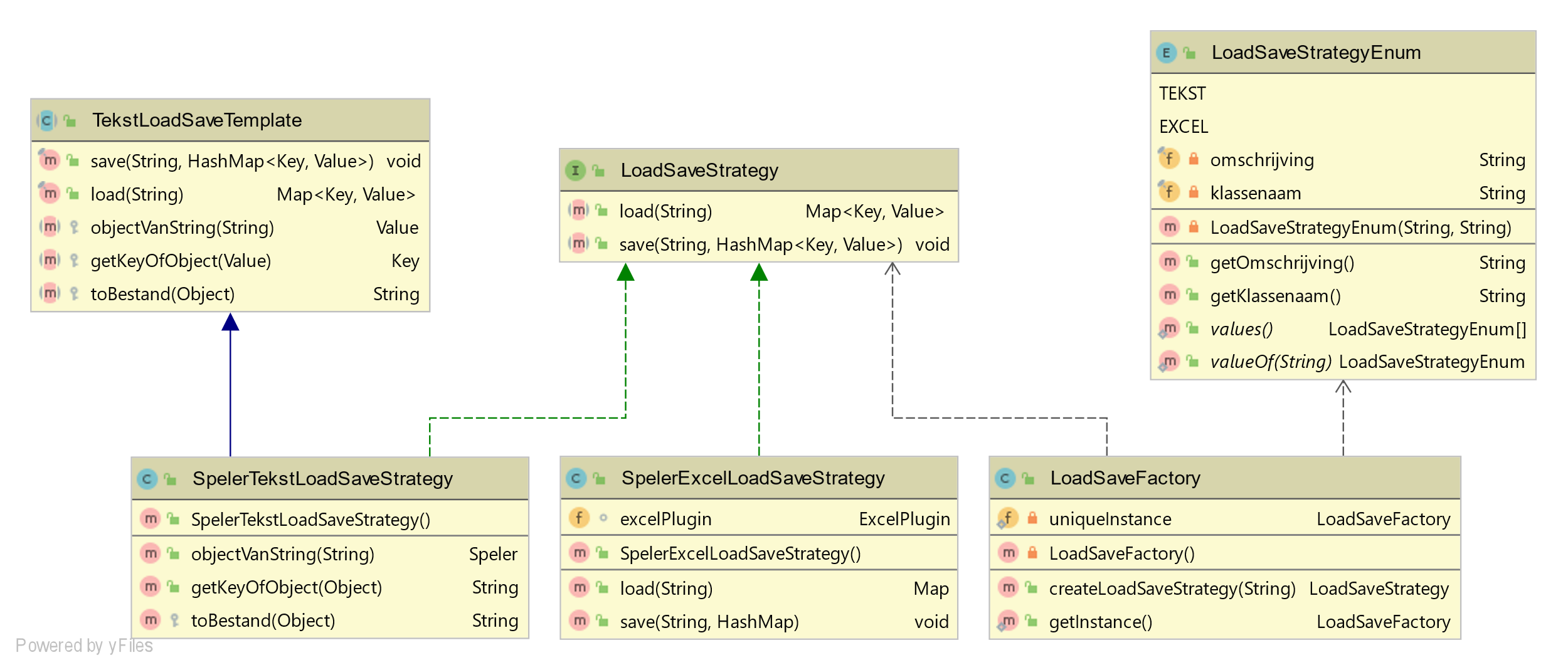
Helper:



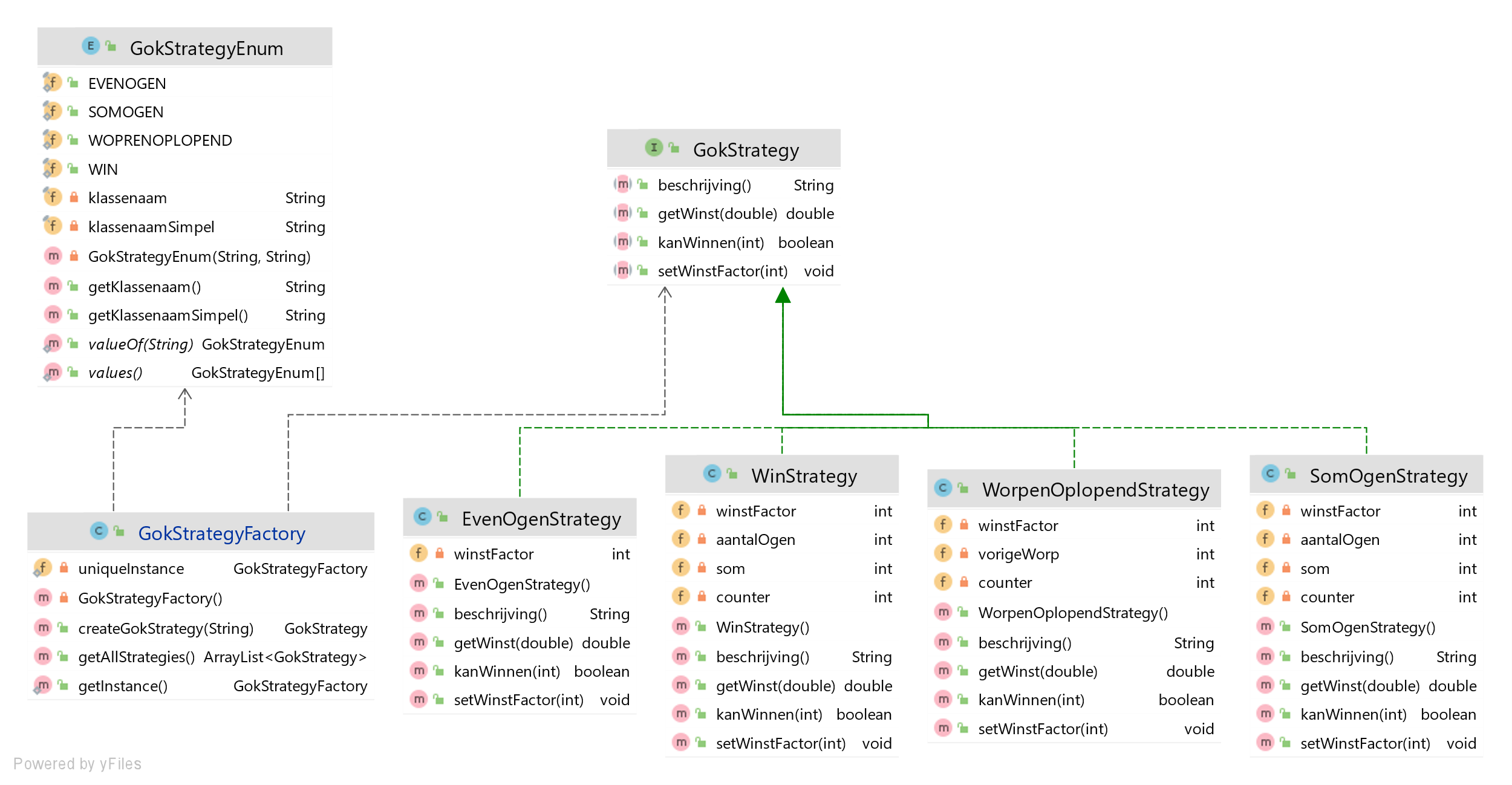
Model > Database:



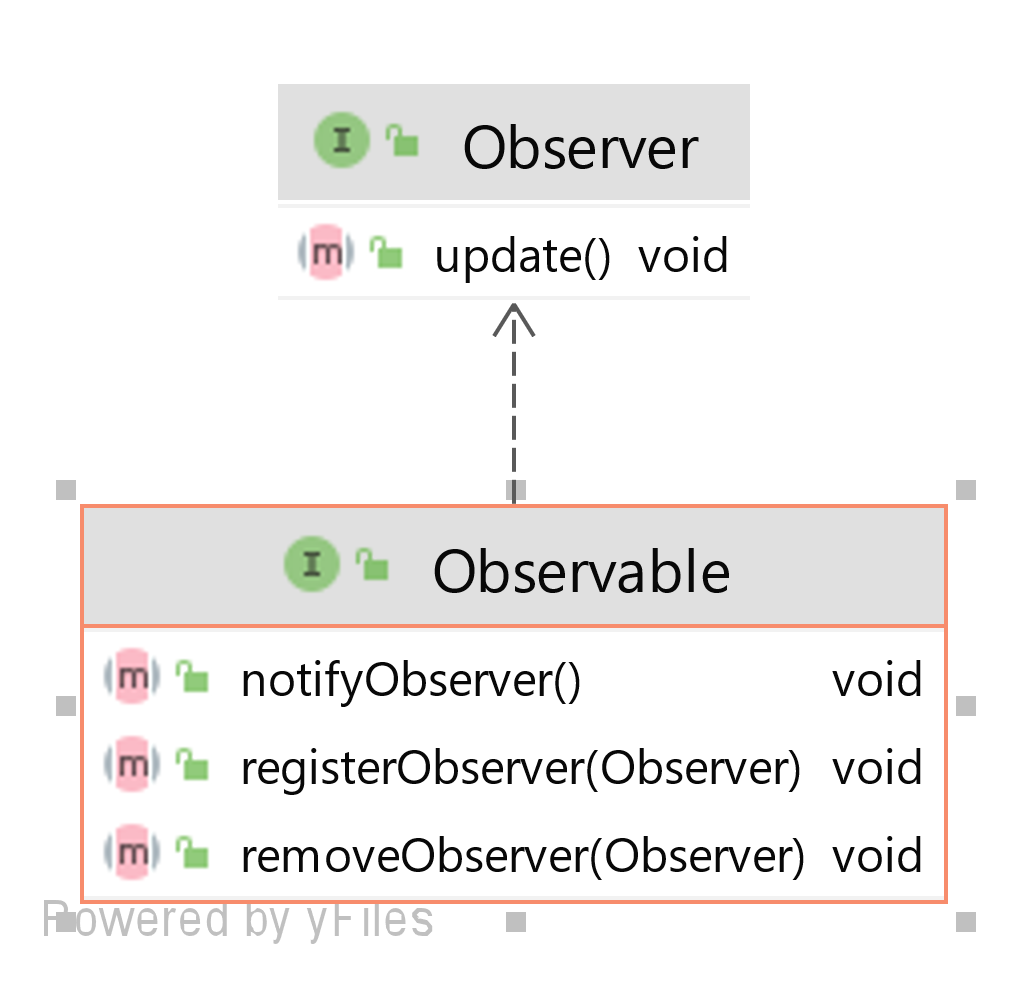
Model > Database > LoadSaveStrategy:



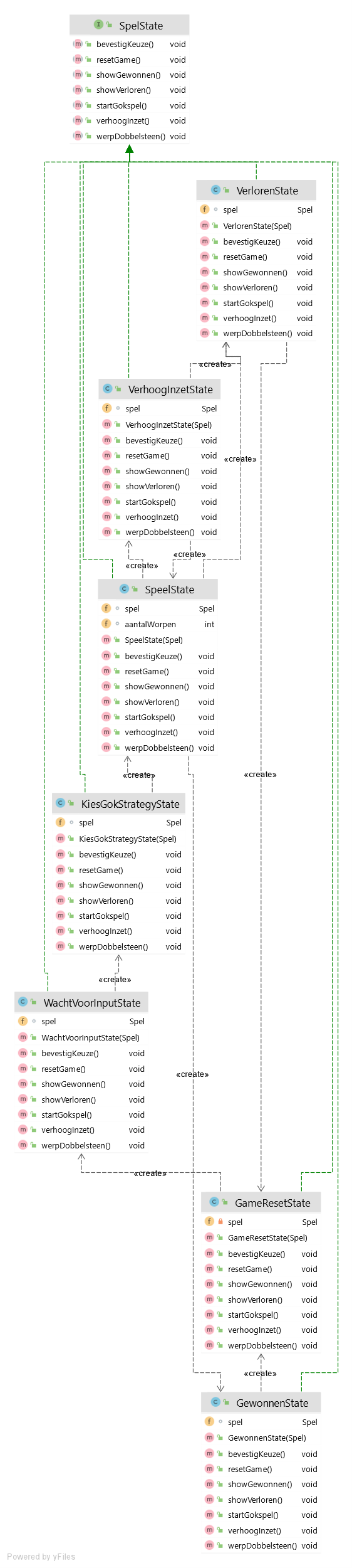
Model > Gokstrategy:



Model > Observer:



Model > State:



View > Panels:

